AD-A095 070 HONEYWELL SYSTEMS AND RESEARCH CENTER MINNEAPOLIS MN F/G 14/2
REAL-TIME FEASIBILITY FOR GENERATION OF NONLINEAR TEXTURED TERR--ETC(U)
JAN 81 D SOLAND, M VOTH, P NARENDRA F33615-77-C-0073
UNCLASSIFIED AFHRL-TR-79-27 NL

1 OF 2
AD A
095070

070

AFHRL-TR-79-27

AIR FORCE

HUMAN RESOURCES

AD A095070

LI

REAL
OF

LA

AIR FORCE
BROOKS AI

NOTICE

This final report was submitted by Honeywell, Systems and Research Center, 2600 Ridgway Parkway, Minneapolis, Minnesota 55413, under contract F33615-77-C-0073, Project 1958, with Operations Training Division, Air Force Human Resources Laboratory (AFSC), Williams Air Force Base, Arizona 85224. Mr. Michael R. Nicol was the Contract Monitor for the Laboratory.

MARTY R. ROCKWAY, Technical Director
Operations Training Division

RONALD W. TERRY, Colonel, USAF
Commander

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER AFHRL-TR-79-27 — 2. GOVT ACCESSION NO. AD-A095	3. RECIPIENT'S CATALOG NUMBER 070
4. TITLE (and Subtitle) REAL-TIME FEASIBILITY FOR GENERATION OF NONLINEAR TEXTURED TERRAIN	5. TYPE OF REPORT & PERIOD COVERED Final
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Duane Soland, Mark Voth, Pat Narendra	8. CONTRACT OR GRANT NUMBER(s) F33615-77-C-0073
9. PERFORMING ORGANIZATION NAME AND ADDRESS Honeywell, Systems and Research Center, 2600 Ridgway Parkway, Minneapolis, Minnesota 55413	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 63227F 19580307
11. CONTROLLING OFFICE NAME AND ADDRESS HQ Air Force Human Resources Laboratory (AFSC), Brooks Air Force Base, Texas 78235	12. REPORT DATE January 1981
	13. NUMBER OF PAGES 114
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Operations Training Division, Air Force Human Resources Laboratory, Williams Air Force Base, Arizona 85224	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Image Generation (CIG)
visual simulation
textured terrain simulation
bicubic splines

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This study was conducted by Honeywell for the Air Force Human Resources Laboratory (AFHRL) in order to evaluate and study a potential new approach for the simulation of visual and sensor imagery for Air Force training applications. This report describes the hardware implementation of a curved-surface method for computer image generation (CIG) of textured terrain imagery. General comments and details of the algorithms are presented. This is followed by a discussion of the hardware required for a real-time implementation of this technique.

DD FORM 1 JAN 73 1473

Unclassified 49

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Item 20 Continued:

The approach involves the display of terrain as curved surfaces represented by bicubic splines. Texture patterns may then be mapped to these terrain surfaces. Buildings or man-made features may be drawn using polygonal surfaces.

This curved approach is of interest because it may represent a more cost-effective method to include more detail in the simulated imagery. Current systems are constrained to the use of straight edges in the representation of real-world features and require large numbers of edges to display complex, irregular objects such as terrain. Therefore, the curved surface approach may demonstrate many advantages over the straight edge technique.

CONTENTS

CONTENTS (concluded)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (continued)

LIST OF ILLUSTRATIONS (concluded)

LIST OF TABLES

SECTION I

INTRODUCTION

The purpose of this study was to determine the real-time hardware requirements for computer generation of nonlinear textured terrain imagery. The approach is based on algorithms and software developed by Honeywell under an Independent Research and Development (IR&D) program. This software was implemented on the Air Force Human Resources Laboratory (AFHRL) Simulation Training and Advanced Research System (STARS) computer facility for nonreal-time generation of imagery during Phase I of this program. The quality of the resulting imagery was considered sufficient to warrant an investigation of real-time hardware requirements during Phase II. This report discusses that investigation.

This effort is part of a larger AFHRL program to apply computer image generation (CIG) to techniques for use in visual and electro-optical (E/O) sensor simulation systems for trainer systems. Emphasis on low-level flying and the need to incorporate training for target detection and identification tasks places new requirements for realism in these simulations. Previous studies have established the applicability of CIG technology to visual and E/O sensor simulation and have developed algorithms for the simulation of atmospheric effects and sensor transfer characteristics. These studies were based on existing CIG techniques which use polygonal representation of surfaces. Smooth shading techniques were incorporated for added realism.

Polygon methods are not readily extended to include texturing of surfaces. Furthermore, the computational requirements for polygonal terrain representation would be considerable. Although smooth shading can reduce or eliminate facetting effects in the interior of objects, silhouettes and boundaries are still composed of straight line segments which detract from the realism of the simulation.

The alternative approach which is the basis for this study represents surfaces as collections of curvilinear, bivariate patches. In general, any surface in three-dimensional space can be represented as a function of two variables. For example, a terrain surface is determined by elevation above some datum. The two independent variables represent coordinates in the datum, or ground plane. Other bivariate functions, representing surface material, texture, etc., can be defined with respect to the same coordinates, thus allowing texturing of surfaces.

The particular representation used here for terrain is the bicubic surface patch, a polynomial approach which has recently received extensive development for use in computer-aided design of surfaces. The bicubic patch representation has first or second derivative continuity across patch boundaries, depending on the particular bicubic representation used (see Appendix A). Consequently, smooth shading and curved silhouettes and boundaries are inherent in the approach.

The generation of a perspective view of a surface consisting of bicubic patches, as required for visual and E/O sensor simulation, requires different methods than those used for polygon surfaces. The method used here is the bicubic subdivision method of E. Catmull.[1] Catmull's method is a fast, recursive subdivision of the patch into smaller and smaller subpatches, until each subpatch is small enough to be displayed. Because the order in which the subpatches are ready for display is random, the usual scanline approach in which the imagery is produced a line at a time in synchronism with the raster scan of the display is not applicable. Rather, the image is assembled in a special memory called a frame buffer.

The frame buffer contains the entire image at display time and is scanned out in raster sequence. Catmull extended the frame buffer concept, originally proposed by M. Newell,[2] to include the distance from the viewpoint to each displayed point. The extended frame buffer is called a Z-buffer, which provides a simple solution to the hidden surface problem by resolving conflicts based on distance from the viewpoint.

The Z-buffer also includes the following advantages. Computing requirements increase only linearly with scene complexity. For other priority algorithms, computing costs increase faster than linearly with, say, the number of edges required for the scene. Also, different and more suitable object representations may be used within the same system. For example, bicubic patches may be used for terrain, polygons for buildings, and quadratics for storage tanks and water towers. Since distance to each picture element is available, atmospheric propagation effects can be included by table look-up at display

[1]E. Catmull, <u>A subdivision algorithm for computer display of curved surfaces.</u> UTEC-CSc-74-133, University of Utah, December 1974.

[2]Newell, M.E., Newell, R.G. and Sancha, T.L. "A Solution to the Hidden Surface Problem," Proc. ACM Nat. Conf., August 1972.

time. Finally, since the neighbors of each picture element are available, two-dimensional interpolation and a form of anti-aliasing filtering are easily incorporated.

In order to study the Honeywell approach to CIG, AFHRL at Wright-Patterson AFB contracted with the Honeywell Systems and Research Center for a software implementation of the algorithm for evaluation. Under Phase I of contract No. F33615-77-C-1173, "A Study of Real-Time Feasibility for Generation of Nonlinear Textured Terrain," Honeywell transferred a working version of its CIG software and data to the AFHRL facility at Wright-Patterson AFB, Ohio. Under Phase II of this contract, Honeywell defined the real-time hardware organization required to implement the algorithm. The study established the requirements using a combination of analytical methods and computer simulation, analyzed several system configurations, and performed a detailed study of memory and computational devices required. This report covers the technical effort for Phase II of the contract.

SECTION II

PROBLEM AND APPROACH

The Honeywell CIG approach combines several innovative ideas. These include using a frame buffer and depth buffer to assemble the components of a scene, using curved surfaces instead of polygons for terrain representation, applying texture patterns to the surfaces, and using polygons to represent cultural objects in the same scene.

The basic elements of the CIG problem for terrain are shown in Figure 1. They consist of a viewpoint represented by an eye, a "window," the terrain, and a line of sight (LOS). The window may be an actual window in the case of visual simulation, or the focal plane in the case of a camera such as forward-looking infrared (FLIR) or low-light-level television (LLLTV) camera.

In simulator visual systems, the window is replaced by a cathode ray tube (CRT) or similar display device with suitable optics to provide a virtual infinity image at the viewpoint. The scene viewed through the window is synthesized by computer from a data base. The data base consists of terrain elevation samples and descriptors of the material at the surface of the ground as a function of geographical coordinates.

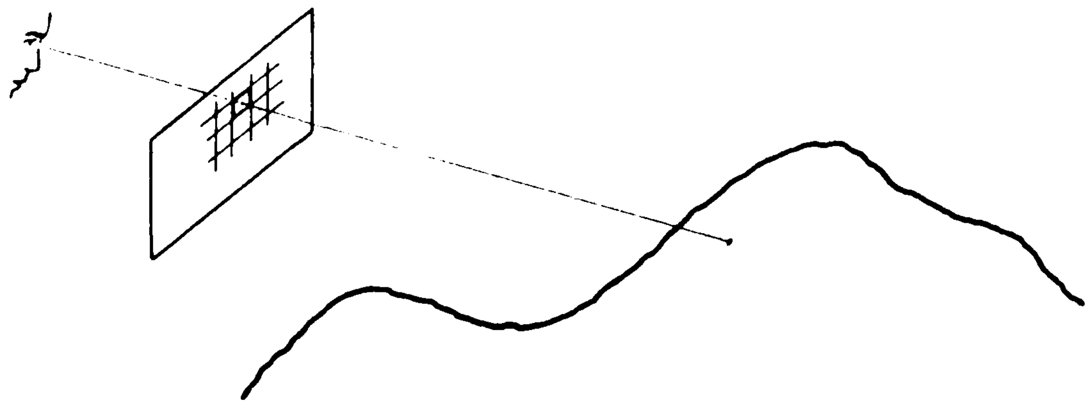


Figure 1. Terrain Image Generation Problem

The synthesized image consists of a rectangular array, or raster, of intensity samples. The intensity or color displayed at a particular raster element is determined by the material characteristics (reflectance, emittance, temperature, etc.) of the first surface intersected by the LOS from the viewpoint through the raster element. It is also determined by illumination, surface orientation, and attenuation along the LOS. For FLIR and LLLTV, the displayed intensity depends also on camera resolution, sensitivity, and noise characteristics.

The raster element should not be considered as a point, however, but as a small element of area on the display. The solid angle determined by the viewpoint and the raster element will intersect the terrain over some patch of terrain (Figure 2). The patch is the projection of the raster element onto the terrain surface. The area of the patch will vary with distance and obliquity of the surface.

The displayed intensity is determined by the average brightness over the path and not by the brightness at a point, as suggested by Figure 1.

Required, then, are algorithms for sampling for average brightness of the ground patch corresponding to each raster element within the field of view. As the viewpoint and orientation of the window change, the projection of ground patches on the display window will change accordingly. The algorithms must be computationally simple enough to allow for real-time tracking of these changes with realistic hardware speeds.

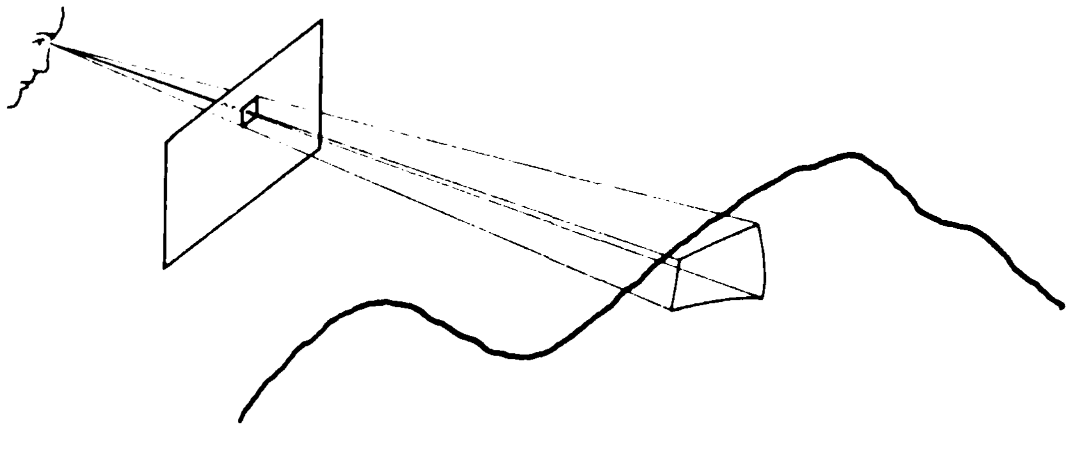


Figure 2. Area Sampling

BICUBIC SPLINES

Bicubic splines provide a smooth polynomial interpolation of the terrain elevation samples in the Defense Mapping Agency (DMA) source data. Each spline approximates a patch of terrain overlaying a rectangular area in the ground plane (Figure 3). The ground plane in this sense is an arbitrary horizontal datum plane. A Cartesian coordinate system is defined with the Z axis aligned vertically and the X and Y axes aligned east and north, respectively, with origin directly beneath the aircraft in the ground plane. The terrain patches then correspond to the latitude/ longitude grid of the DMA source data.

The patch boundaries are intersections of the four vertical planes along the latitude/longitude grid lines with the terrain surface, and are usually curved due to the bicubic representation. The projections of these curves onto the window are also curved. Consequently, unlike polygon representations, the four vertices cannot simply be connected with straight lines in the window plane to determine the imaged patch boundaries. The approach taken here is to subdivide the patch until the resultant subpatches each cover one picture element.

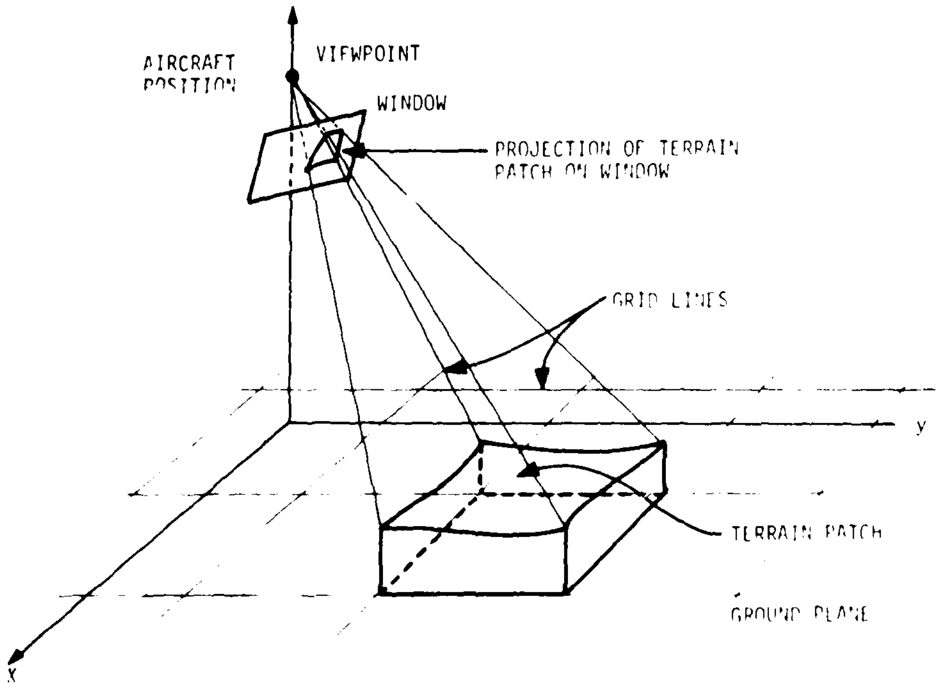


Figure 3. Terrain Imaging Geometry

A bicubic spline provides a polynomial representation of terrain elevation z of third degree in x and y which can be written as

$$z = a_o + a_1 x + a_2 y + a_3 x^2 + a_4 y^2 + a_5 xy^2 + \ldots + a_{15} x^3 y^3$$

A bicubic has 16 arbitrary coefficients which are determined by a minimum of 16 terrain elevation samples. The simplest approach uses the four corner elevations for the patch and the neighboring 12 elevation samples.

The bicubic coefficients can be simply determined by arranging the 16 elevations into a 4 x 4 matrix $\underset{\sim}{Z}$ and then pre-multiplying and post-multiplying $\underset{\sim}{Z}$ by a 4 x 4 matrix of constants, $\underset{\sim}{M}$, and its transpose, respectively. Appendix A discusses bicubic spline mathematical properties in more detail.

Depending on the choice of $\underset{\sim}{M}$, a bicubic may be either <u>interpolating</u> or <u>approximating.</u> An interpolating spline surface, which has first derivative continuity, is one which fits the true surface at the four corners of the patch and interpolates the surface elsewhere. An approximating spline, which has second derivative continuity, approximates the surface everywhere.

In either case, the resulting surface is smooth with at least first derivative continuity everywhere, including along patch boundaries. Thus, the normal to the surface is continuous. This is important because shading of surfaces depends on the angles between the surface normal and lines to the viewpoint and illumination source. The human eye is very sensitive to spatial discontinuities in shading, even in the first derivative of shading; therefore, discontinuities in the surface normal may be discernible. The effect might be particularly objectionable, and in fact may provide false cues to the trainee, in real-time visual simulation, if the discontinuities appear to jump or crawl as the angles change.

Special interpolation techniques have been developed to reduce these effects when representing curved surfaces by polygons. Examples are Gouraud[3] shading and Phong[4] shading. An advantage of bicubic spline representation of surved surfaces is that the necessary continuity is inherent in the approach. No smoothing approximations are necessary.

[3]Gouraud, H. "Computer Display of Curved Surfaces," Dept. Comp. Science, U. of Utah, Tech. Rept. UTEC-CSc-71-113, June 1971.

[4]Phong, Bui Tuong, "Illumination for Computer-Generated Images," Dept. of Comp. Science, U. of Utah, Tech. Rept. UTEC-CSc-73-129, July, 1973.

Bicubics are the splines of lowest degree which have this property. Although biquadratic splines require fewer coefficients, they cannot provide the desired degree of continuity. This is the principal reason bicubic splines were chosen for terrain representation.

GROUND PLANE INTENSITY REPRESENTATION

The previous subsection dealt with the mathematical representation of terrain for CIG. The terrain can best be represented by bicubic spline functions; these are bivariate polynomials determined by the data base terrain elevation samples. The bicubic representation was chosen for its smoothness properties.

The other half of the DMA data base consists of planimetric features which can be represented as a map in the ground plane (Figure 4). Each surface material (concrete, grass, rock, sand, etc.) is represented by a code as a function of latitude/longitude grid coordinates. From this data, a map like that of Figure 4 can be constructed to the resolution level of the data base. Such a map can also be made from any two-dimensional intensity or color distribution such as a photograph or sensor image. Aerial photographs and random patterns were used in the Phase I software implementation.

It is possible to assign an intensity (representing reflectance, emittance, or other material characteristics) to each surface material. Each intensity can be mapped from the ground plane to the image plane by mapping the curved terrain surface onto the image plane (Figure 5). The mapped intensity value can then be modified for illumination, temperature, atmospheric attenuation, noise, etc. to give the displayed intensity.

The ground plane intensity values can be specified to any desired spatial resolution. In particular, this concept allows texture to be specified in the ground plane for each terrain material and mapped onto the image plane with correct perspective, occlusion, and variable resolution with distance.

Texture in images can be described as a quasi-periodic or random variation of intensities which gives a surface a particular character. The surface is then perceived as tree-covered, or as a plowed field, etc., by the viewer. Perspectively correct texture can also give the impression of distance, surface slant, and motion. Consequently, texture should add significantly to realism and depth and motion cues in CIG.

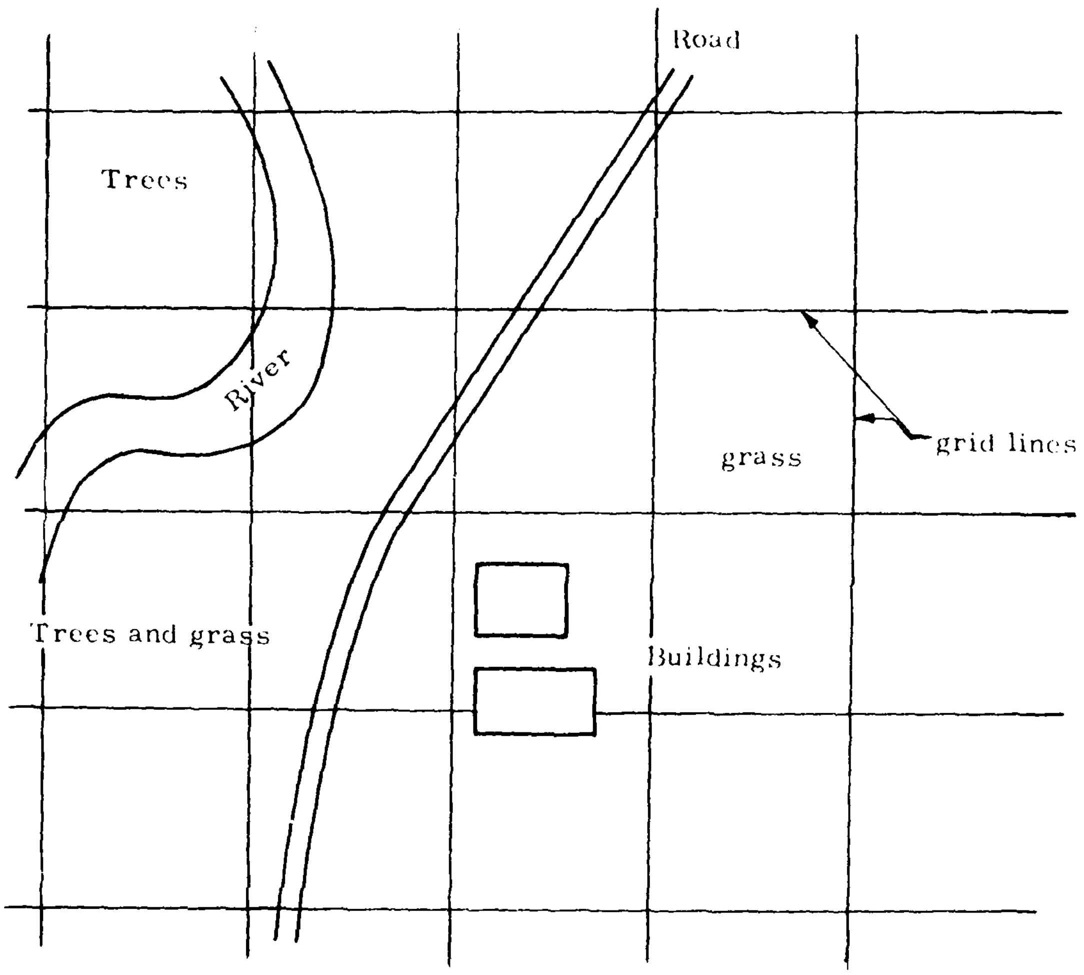


Figure 4. Ground Plane Map of Planimetric Features

BICUBIC SURFACE MAPPING

The approach described here for CIG using bicubic spline terrain representation is based on Catmull's bicubic spline subdivision algorithm. The algorithm is described in detail in Appendix B.

Figure 6 illustrates a mapping of a patch onto the image plane and two successive stages of subdivision. The original patch corners are denoted A, B, C, D. The projection of the patch boundary curves are shown connecting these points. Catmull's algorithm provides the new boundary curves for the binary subdivision of the first patch.

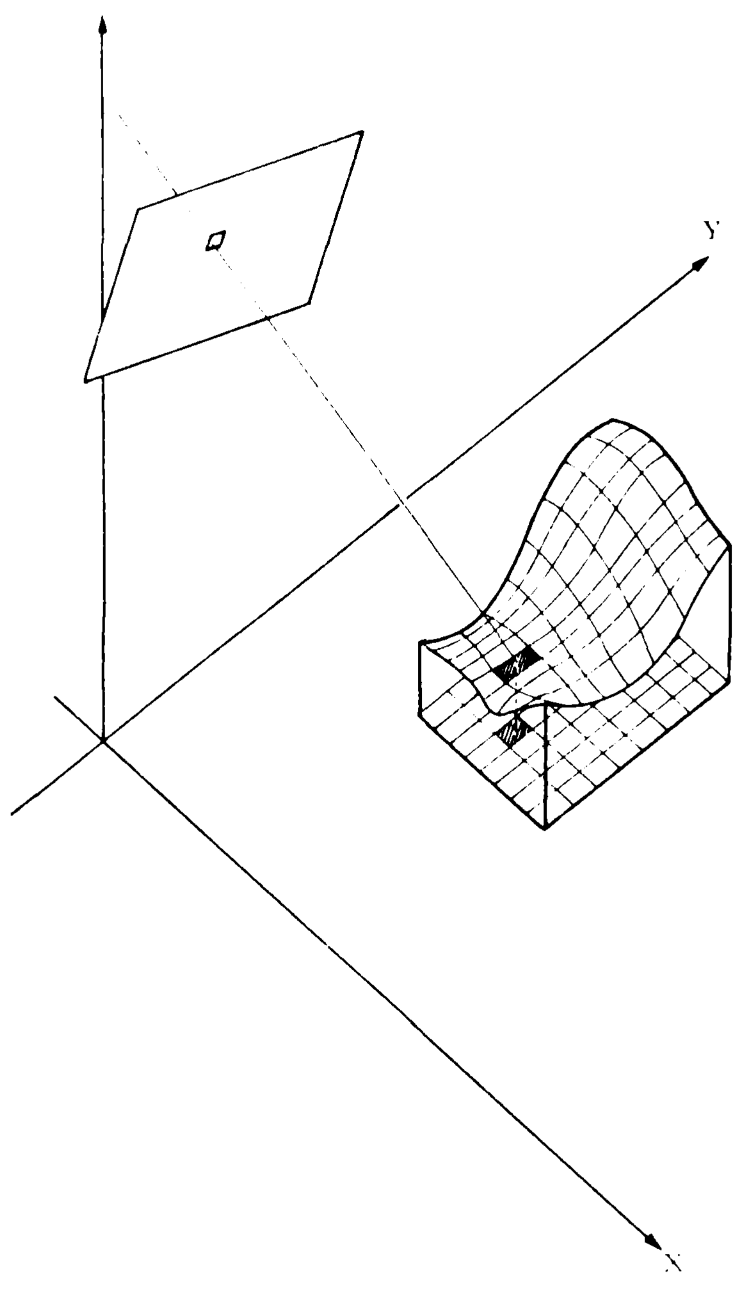


Figure 5. Mapping of Ground Plane Intensities onto Image Plane

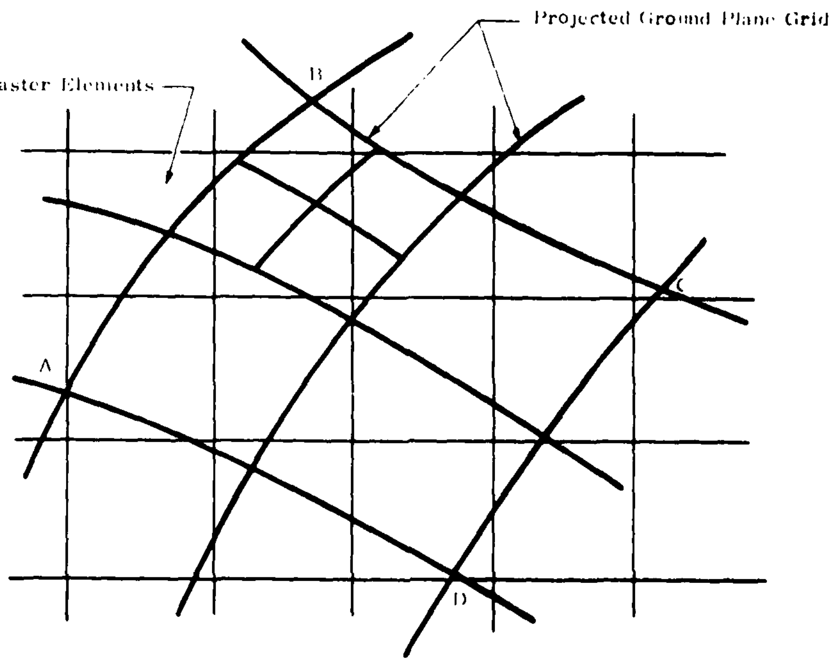


Figure 6. Mapping of Ground Plane onto Image Plane

Each bicubic patch is successively subdivided into four subpatches until each of the resulting subpatches, when projected onto the image plane, occupies the area of one raster element. The visibility of each such subpatch is determined and the corresponding intensity displayed. Because the image is constructed in a scattered fashion, a frame buffer is used to assemble the picture prior to display. If distance to and intensity of each subpatch is stored in the buffer for each raster element, the visibility determination is greatly simplified. This concept is called a Z-buffer.

Several sequences of scenes have been generated to demonstrate the capabilities of the program. Figure 7 shows successive views of one corner of a data base including some buildings on the terrain. Figure 8 is a series of views of a more mountainous portion of the data base.

The data base environment for Figures 7 and 8 consists of a hilly area of southwest Nevada. The rectangular region represented is about 5 km per side with the origin at the southwest corner of the rectangle. This region can be seen on the U.S. Geological Survey (USGS) topographic map, Pahramp quadrangle. The origin is at approximately 115 deg 53 min W, 36 deg 12 min N. Figure 9 shows the boundary of the data base area.

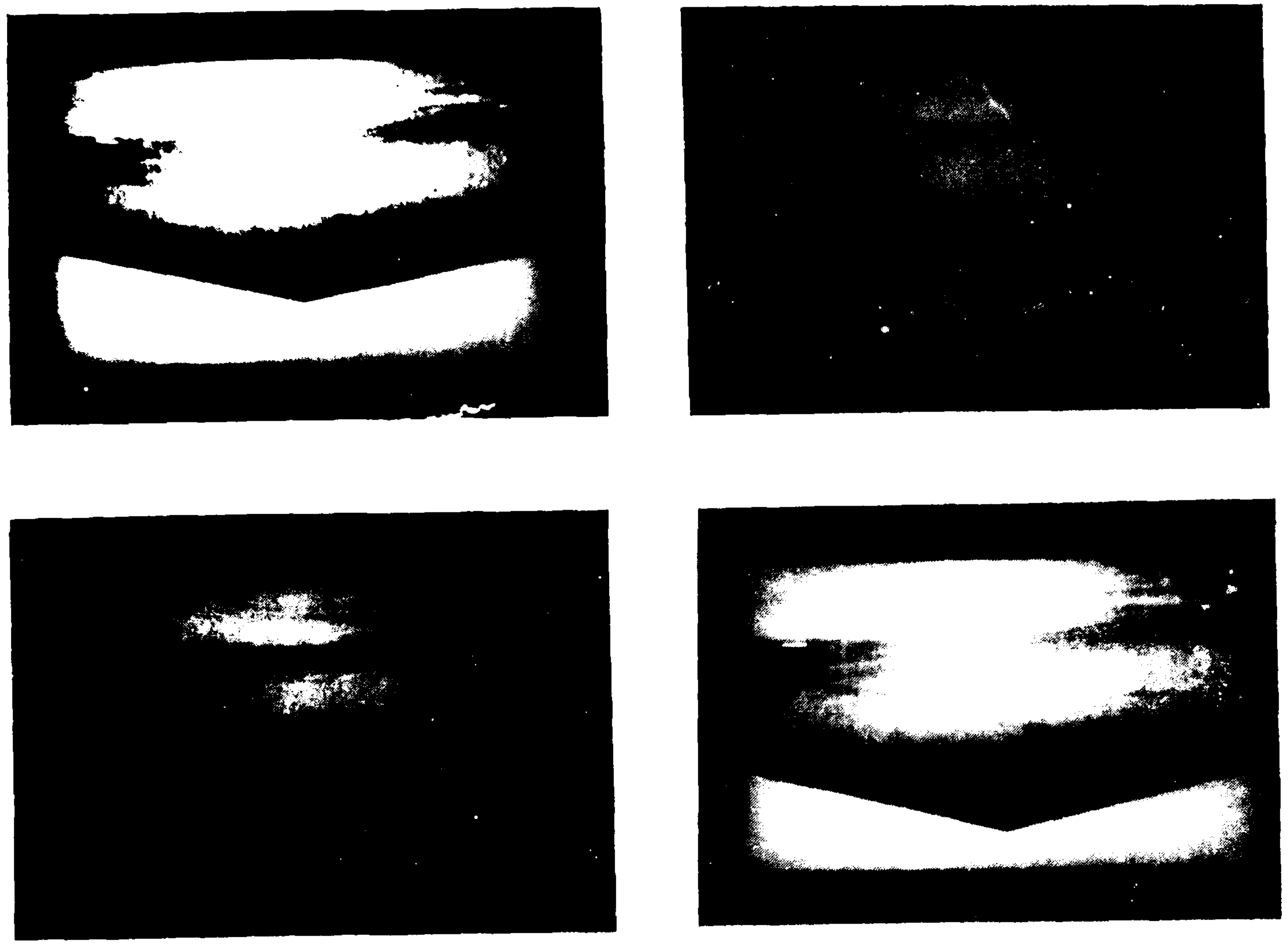


Figure 7. Nonlinear Textured Terrain Sequence Including Buildings

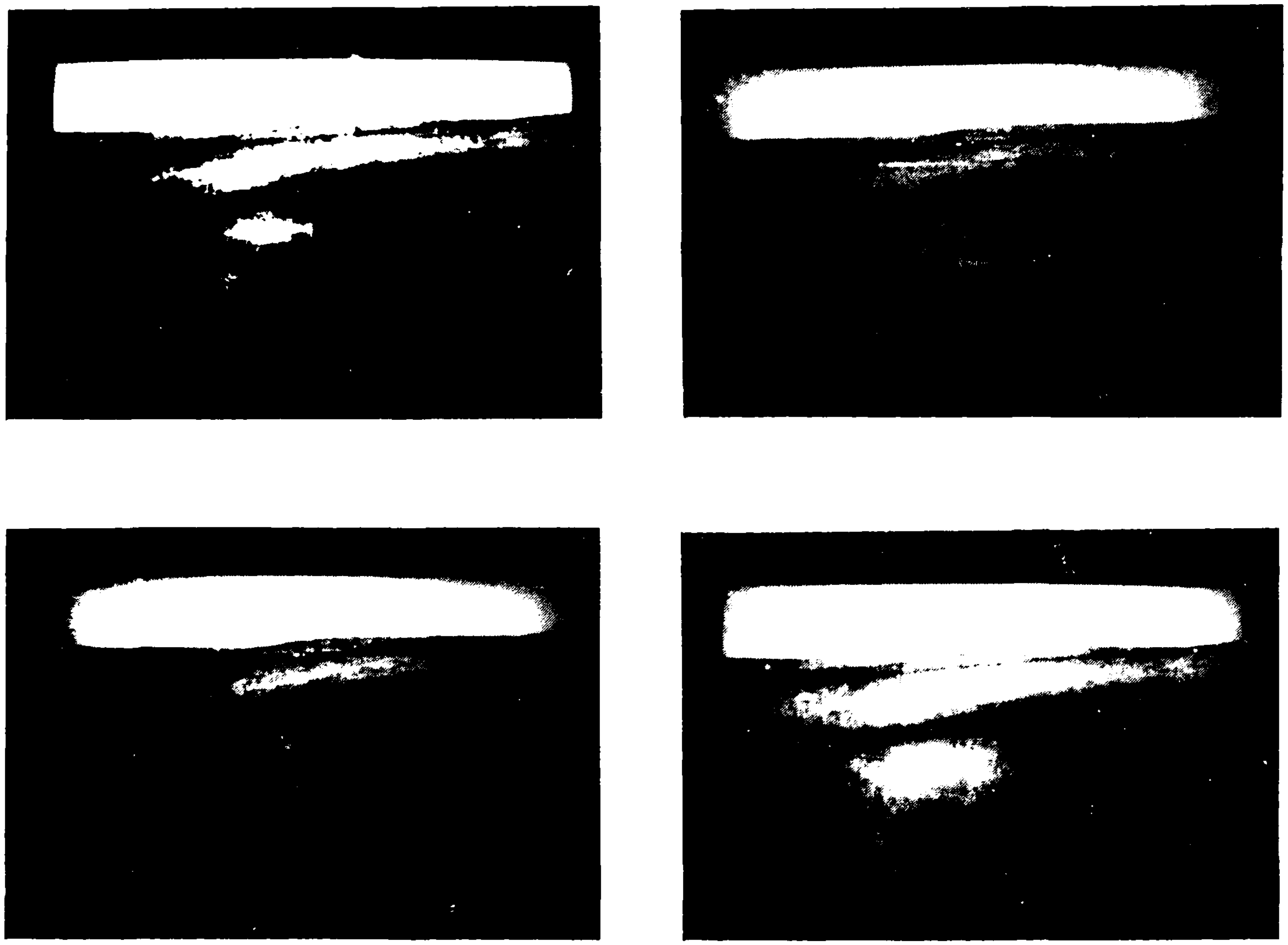


Figure 8. Nonlinear Textured Terrain Sequence

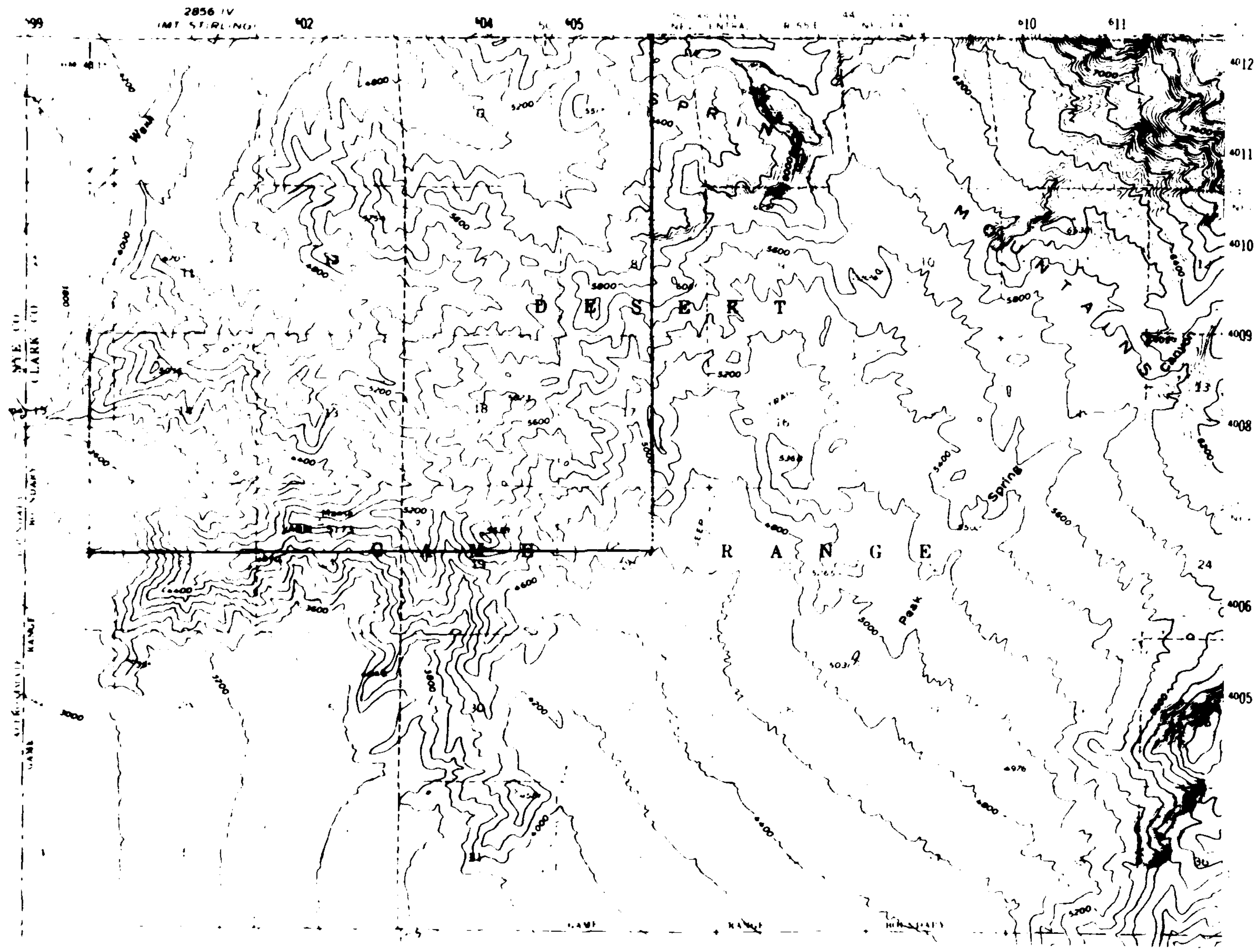


Figure 9. Pahramp Quadrangle, Nevada Series, USGS

SECTION III

SYSTEM CONCEPT

In this section, the algorithms selected for the real-time system feasibility study will be described. Alternative algorithms which were considered are also discussed, along with the rationale for the selection which was made. Functional requirements for the system approach will also be described.

QUAD TREE DATA STRUCTURE

The basic approach is an outgrowth of two ideas. One is to use a Z-buffer to assemble components of a scene. The second is to use a tree structure to represent the terrain data; the root node of the tree represents a square containing the entire geographical area for the mission. Each node below it in the tree represents a smaller area and thus a higher level of detail. Figure 10 illustrates this quad tree structure and the manner in which it represents increasing level of detail in the ground plane.

The algorithm involved in creating a picture is a tree scan. It is not a tree search for some particular predetermined elements but a scan of as many branches as are needed to find those nodes which represent each pixel on the display screen. Of course, a data structure containing all possible pixels is much too large to implement, so the lowest nodes in the tree, the terminal nodes or leaves, are representations of curved surface patches.

The ground area represented by each terminal node is determined by the resolution inherent in the terrain elevation data base. For the scenes depicted in Figures 7 and 8, each terminal node represents a ground patch of approximately 100 meters on a side. Parents of the terminal nodes represent patches of four times as much area, the next level of nodes 16 times and so on up the tree. Thus, each node represents a quarter of the area in the ground plane represented by its parent node.

Each node in the tree carries the following data about the surface area it represents:

- Flag to indicate terminal or non-terminal node
- X and Y coordinates of a specified corner of the patch
- Length of the sides of the patch

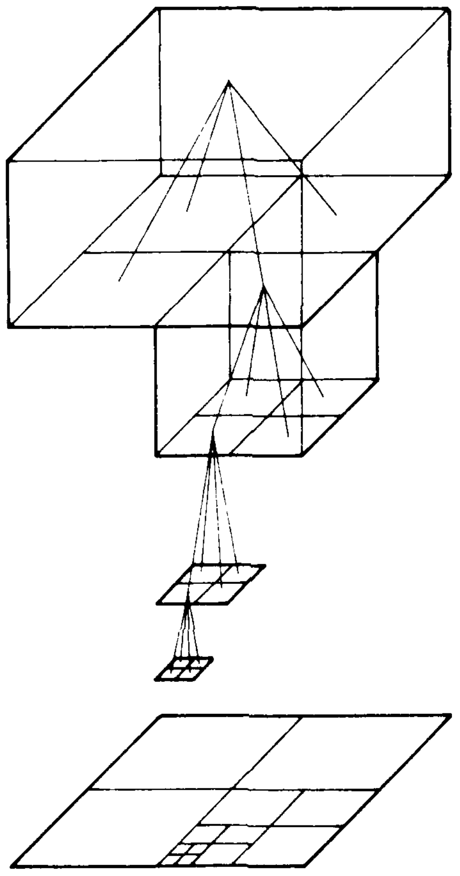


Figure 10. Quad Tree Data Structure

Non-terminal nodes also carry

- Elevations at each of the four corners of the patch
- Intensity (average of the intensities of the four subnodes)
- Pointers to its four subnodes

Terminal nodes carry

- A 16-entry array which is a bicubic representation of the surface
- A texture map for the surface. The map is an $2^k \times 2^k$ array of M bit intensities (for Figures 7 and 8, $k = 3$ and $M = 3$)

The tree is scanned in depth first order. That is, the algorithm steps through the three from node to node, beginning at the top, or root node. Nodes near the top represent large ground areas whose projections on the display screen will depend on distance from the viewpoint. Projections of nearby nodes will cover many pixels, so the tree must be scanned to greater depth for nearby nodes. Distant nodes will project to smaller picture areas and the scan depth will be less. This feature provides an automatic level of detail selection as well as minimizing the computational effort in producing a picture.

The procedure for generating a picture, then, is as follows. Starting with the top node, the picture area corresponding to the node is determined by projecting the four corners of the patch represented by the node to display screen coordinates. A decision is made to terminate the scan at that node or proceed to the next level. Termination can occur under the following conditions:

1. The projected area represented by the node falls outside the display window.
2. The projected area faces away from the viewpoint.
3. The projected area is small enough to display.

The result of any of these conditions is to terminate the scan at that node and discard all branches and nodes below the termination node. When the entire tree has been scanned in this manner, the result is the identification of all potentially visible nodes, each node representing a pixel-sized area on the display screen, and the corresponding display coordinates. However, some of the potentially visible nodes will be occluded by other nodes. The priority, or which nodes are actually visible, can easily be determined by comparing the distance from the viewpoint to each node where priority conflicts occur.

PRIORITY CONFLICT RESOLUTION

The resolution of priority conflicts uses a combination of a frame buffer and depth, or distance, buffer called a Z-buffer. The frame buffer holds the intensities or tones of display points as they are generated. The depth buffer stores the distances of these same points and resolves priority conflicts by keeping only the nearest point (both intensity and distance) when conflicts occur. An added benefit of this approach is that when the frame is displayed, the distance to each pixel is available for intensity modification due to haze, fog, or other atmospheric attenuation.

SCENE GENERATION

The first step required to produce a picture or a sequence of pictures of the terrain is data base preparation. This step consists of building the data tree structure with the data elements described above. This step may be considered to be a restructuring of the DMA terrain elevation samples, augmented by texture data. The texture samples should conform to the DMA cultural file to represent the various terrain materials and their boundaries. Since the scene generation algorithm under unvestigation in this study does not depend on how the texture arrays are actually generated, details will not be described here. The textures in Figures 7 and 8 were produced by digitizing aerial photographs of the terrain being represented. This technique is only one of several that might be used.

The second step is to produce a skeleton tree for the present field of view, depending on viewpoint location, direction and angular extent of the field of view. The skeleton tree is a thinned out tree that contains pointers into the full mission data tree but has been pruned to include only surfaces in the current field of view. This step is repeated for each new frame. It is not necessary, but can substantially reduce the size of the tree to be scanned if the field of view is much smaller than the total mission area. The pruning is accomplished simply by projecting the field of view onto the ground plane and discarding those nodes which are completely outside the field of view boundaries.

The terrain surface gets processed a node at a time, starting with the root node of the skeleton tree. The depth buffer is initialized to a large value. The following steps get repeated until no nodes remain to be examined.

1. Fetch the next node. If none remain, the picture is ready for display. Exit.

2. Project the corners of the node to viewpoint coordinates.

3. Count the number of display points covered by the projected node.

4. If this is a terminal node and there is more than one point covered, subdivide the node (Figures 11, 12) and go to step 1.

5. If there are none, stop searching this branch of the tree and go to step 1.

6. If there is only one and it is behind the viewpoint, discard the node, stop searching this branch and go to step 1.

7. Compare the node distance to that currently stored in the depth buffer at the covered display point. If greater, discard the node, stop searching this branch and go to step 1.

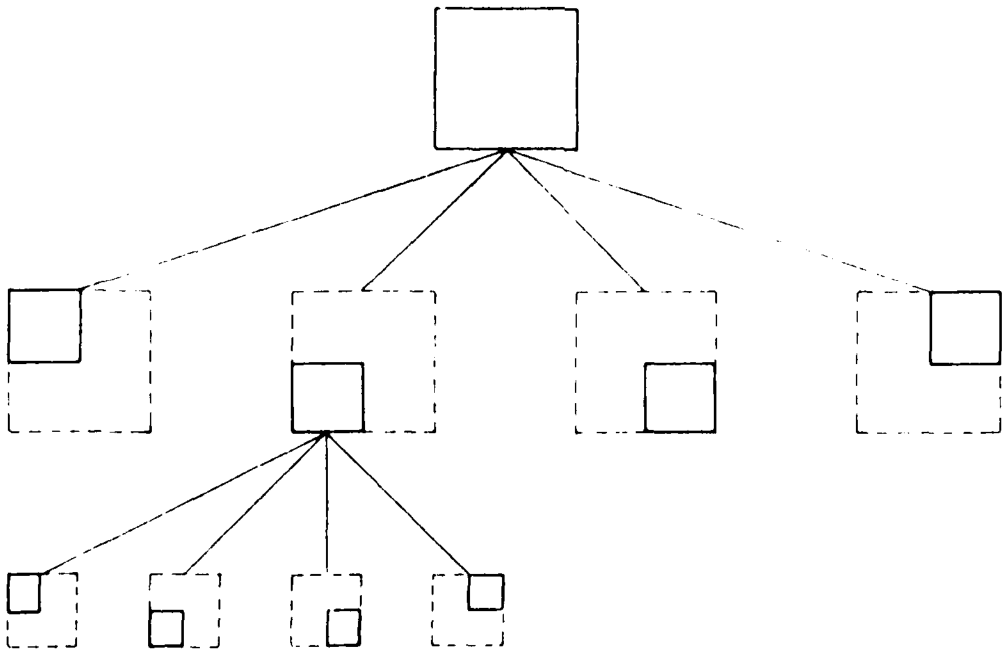


Figure 11. Ground Plane Subdivision Process

8. Compute the displayed intensity for this node, store it in the frame buffer, and store the node distance in the depth buffer.

9. Go to step 1.

This process will execute a depth-first search of the skeleton tree and assures the examination of every relevant node. Any visible surface in the field of view gets written to the frame buffer.

ALGORITHM DETAILS

This subsection describes the algorithms used in Steps 2, 3, 4 and 8.

Projection of Node Corners

The corners of a node are known in ground coordinates as obtained from the DMA terrain elevation file. To determine the position of a patch on the display the four corner coordinates are translated and rotated into the viewer's frame of reference and projected to display coordinates.

Count Display Points Covered

To know whether to stop searching some branch of the tree, the number of pixels covered by the projected patch must be determined. If two or more pixels are covered, the node must be subdivided. The projected patch is approximated with a quadrilateral whose vertices are the four projected corners (Figure 13b). The number of pixel points within this quadrilateral is then counted. Another method is to count the grid points in a "box" determined by the vertices of the quadrilateral (Figure 13c). These methods of determining the size of the projected area are evaluated in the tradeoff studies described in Section IV.

For non-terminal nodes, subdivision occurs naturally as a result of continuing the depth first tree scan. For terminal nodes, a bicubic representation of the patch is maintained known as a register square. Register square subdivision is described in detail in Appendix B. Briefly, the bicubic representation of the four quarters of a whole patch can be generated from the register square of their parent with a small number of shift and add operations. Each quarter can then form a new register square for further subdivision if necessary. Textural information is subdivided in parallel with evaluation.

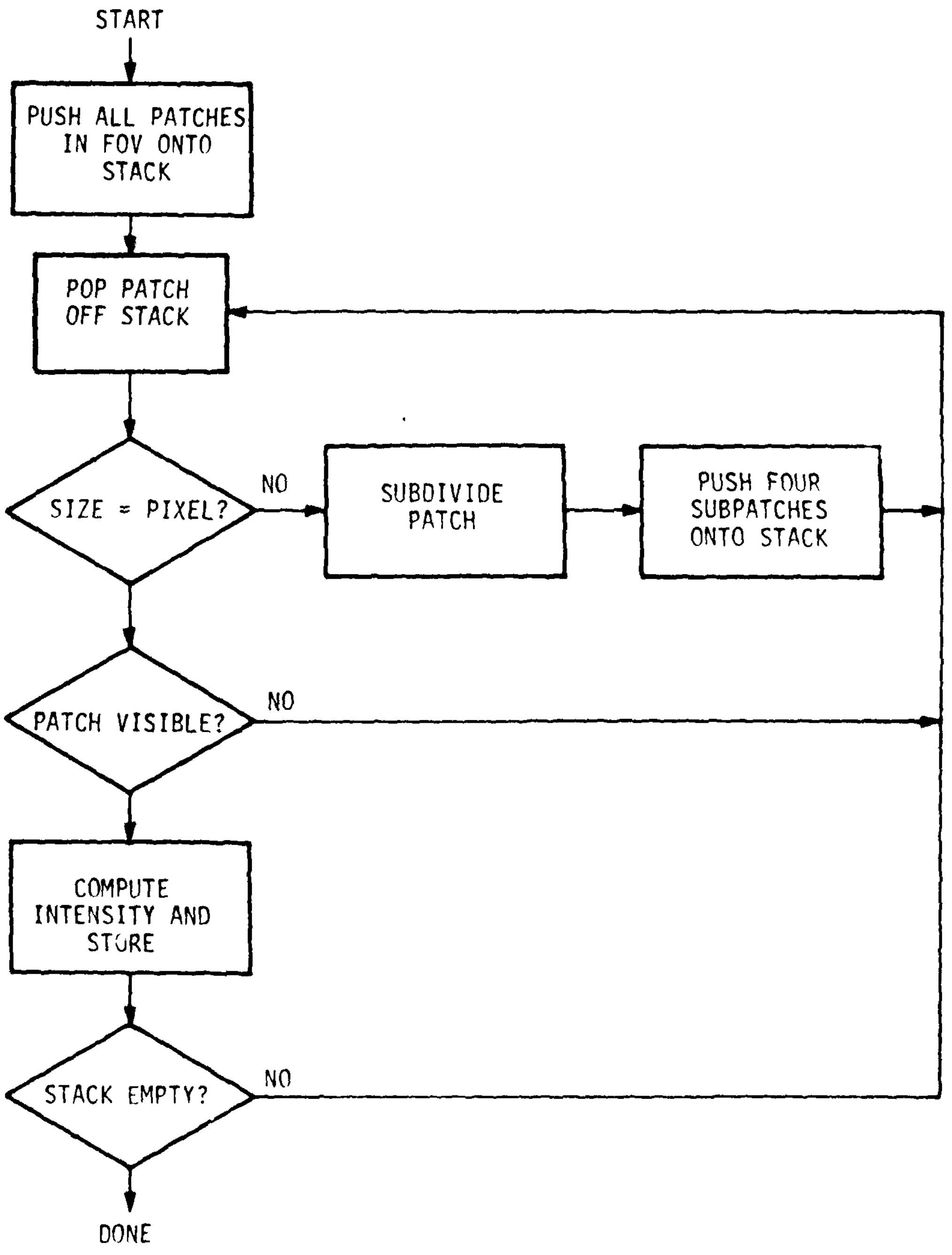

START
PUSH ALL PATCHES IN FOV ONTO STACK
POP PATCH OFF STACK
SIZE = PIXEL?
NO
SUBDIVIDE PATCH
PUSH FOUR SUBPATCHES ONTO STACK
PATCH VISIBLE?
NO
COMPUTE INTENSITY AND STORE
STACK EMPTY?
NO
DONE

Figure 12. Subdivision Algorithm

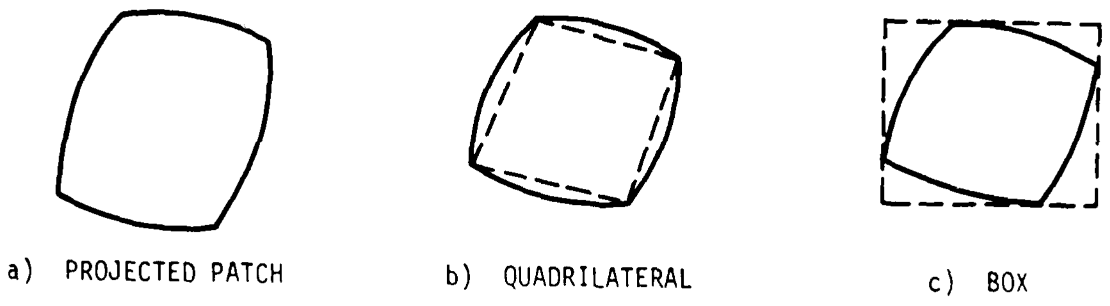


Figure 13. Alternatives for Grid Test

The point intensity calculation uses some preset constants, the textural value of the subdivided patch, the distance of the patch from the viewer, and three vectors: the patch surface normal, the line of sight from the viewer to the patch, and, for visual scenes, a ray from the sun to the patch. Haze is an exponential function of distance. The intensities of the back sides of illuminated areas are thresholded with an ambient intensity constant. The point intensity computation is as follows:

T_p	-	Patch texture value
T_w	-	Texture weighting factor
I_A	-	Ambient intensity
Z	-	Distance of patch from viewpoint
H	-	Haze decay factor
I_H	-	Intensity of haze
$\vec{SN}$	-	Surface normal vector
$\vec{LOS}$	-	Line of sight vector
$\vec{SUN}$	-	Sun vector

R is the dot product of the surface normal vector and the line of sight vector times the dot product of the surface normal vector and the sun angle; this product is thresholded by the ambient intensity.

$$R = \max (I_A, \vec{SN} \cdot \vec{LOS} * \vec{SN} \cdot \vec{SUN})$$

$$T = T_w * R * T_p + R * (1.0 - T_w)$$

$$I = \text{point intensity} = T * e^{-ZH} + I_H * (1 - e^{-ZH})$$

Texture Generation and Representation

The DMA data base specifies 13 classes of texture (surface materials) in various bounded areas (or as a percentage of cover). The generated texture information must be added to enhance the data base. Texture functions can be generated either by actual aerial photographs of a given texture, actual terrain data or by computer synthesis.

Texture Function from Aerial Photographs--The texture function to modulate the diffuse component of the intensity model may be generated with aerial photographs of the texture. It may work well for planar texture only (when only the "pigment" of the surface is textured) as when representing grass, gravel, etc. However, the digitized photograph of a three dimensional (nonplanar) texture (bumpy terrain or trees, for example), fails to simulate the appearance of real texture, because the viewing angle and light source direction of the original photograph is very rarely the same as that of the real-time simulation.

Texture Function from Actual Elevation Data--Texture functions for modulating the surface normal can be derived from physical elevation data just as the aerial photos generated the pigment function above. Examples are terrain wrinkles, bumpy surfaces, trees in clumps, etc. This terrain detail may or may not have any correspondence with the real terrain texture in a given gaming area. The surface normal function can be computed directly from elevation data.

Synthetic Texture Function Generation--Synthetic texture functions have the potential of saving storage (if real-time synthesis is achieved) of the texture function. Texture function synthesis can be either deterministic or probabilistic. A trivial example of deterministic synthetic texture is a "cultural" texture generated by an orderly (predetermined) array of blocks representing buildings. This can be either planar (painted on the surface) or three dimensional and used as the surface normal texture as well. The problem with this is the need to store a large number of different texture maps--otherwise all "towns" would look identical.

Probabilistic texture synthesis allows the random generation of texture subject to a given set of constraints. Simplistically, in the above example, the houses may be required to be in blocks but their spacing and size may be chosen from a random number generator.

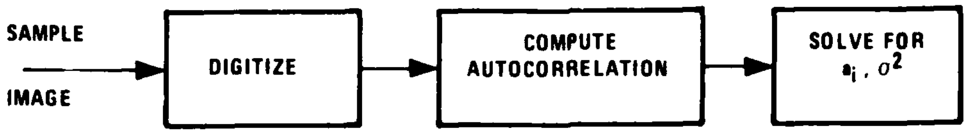
Autoregressive Texture Synthesis--Probabilistic texture synthesis works for natural textures as well (perhaps better than in cultural textures). This is based on the fact that texture functions can sometimes be modeled by an autoregressive function driven by white noise. Honeywell has investigated this Markovian texture synthesis under its ongoing IR&D effort. The block diagram in Figure 14 illustrates this technique. The coefficients of the autoregressive function determine the texture pattern. The coefficients can be determined from a sample of the texture being simulated (say from a piece of an aerial photo). If the texture can be adequately modeled by the autoregressive function, the synthesized texture can look remarkably like the original texture. An additional bonus--the synthesized texture can be of arbitrary extent without repeating itself (as in certain "tiling" schemes).

Tremendous data compaction can be achieved if texture is represented and synthesized this way in real time. The only storage required is for three coefficients for a first order two-dimensional generating function, and the initial values for the filter.

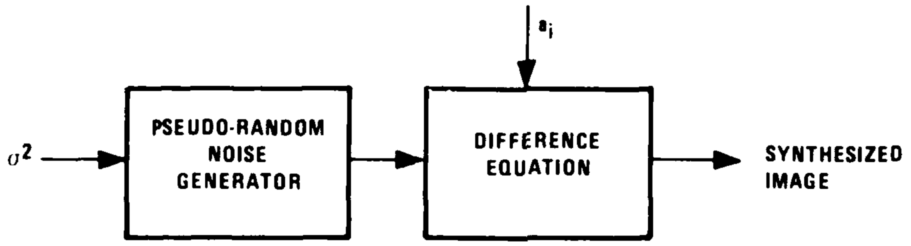
ALTERNATIVES AND TRADEOFFS

Choice of Subdivision Coordinate Systems

The principal choice to be made is whether to subdivide the terrain patches in the ground-based coordinate system or in the moving coordinate system centered at the viewpoint, i.e., in object space or image space. If the subdivision is done in the ground-based coordinates, the resulting subpatch



a) Estimation of the Model Parameters



b) Synthetic Texture Generation Process

Figure 14. Markovian Texture Synthesis

corners must be projected at each node during the tree search. An advantage to this approach is that the register squares can be generated off-line and stored as part of the data base.

The image space approach requires that the register squares be computed for every frame in real time. Only the patch corners are projected, resulting in far fewer multiplication operations. For this reason, the image space subdivision approach was selected. In image space, the terrain is a three-dimensional vector parametric surface, so that three functions must be subdivided simultaneously. For the ground-based approach, only scalar subdivision is necessary.

Another interesting variation would be to perform the subdivisions off-line in the ground-based coordinate system. The tradeoff is clearly one of increased memory requirements for off-line subdivision versus increased computation for on-line subdivision. Since the ground-based subdivision was not selected for the real time feasibility study, this tradeoff was not performed.

Binary Versus Quaternary Subdivision

Quaternary subdivision is the four-way subdivision shown in Figure 11. An alternative is to divide the patch into two subpatches, rather than four, which can be called "binary" subdivision. After projection to the display coordinates, a patch may be elongated since patches become more foreshortened with distance from the observer in one direction (along the LOS) than the other (perpendicular to the LOS). In that case, fewer subdivisions are needed if the patch is always divided along its longest projected dimension. This tradeoff was done and results are described in Section IV.

SECTION IV

SYSTEM DESIGN

REAL-TIME SYSTEM APPROACH

The primary objective of the study was to investigate the feasibility of implementing the bicubic spline subdivision approach in real time for simulators and similar applications. This was done by performing a detailed requirements analysis and system design at the functional level.

Figure 15 shows the basic functions required in the real-time system. Data for the simulated mission, consisting of terrain elevations and ground plane texture are stored on some mass storage device such as disc. As the aircraft moves over the terrain, data for the patches within the field of view (FOV) of the aircraft are transferred to the FOV memory. The FOV memory can be visualized as a window which moves across the terrain with the aircraft. The size of the window is determined by the range of maximum visibility.

The FOV memory is organized as a toroid with a relatively slow input port and a high speed output port. For each frame, the contents of the FOV memory are scanned and processed to generate the displayed image.

The first processing steps are referred to as the Register Square Processor. This processor provides the following functions:

- Transforms terrain data from geocentric to eye-centered coordinates with x and y axes parallel to the display x and y axes
- Projects terrain samples onto image plane
- Determines which patches require subdivision
- Computes bicubic register square values for each patch in the FOV

The register square values for each patch are the data necessary for patch subdivision. These are stored in a buffer memory which is organized as a stack.

The contents of the stack are subdivided by the Patch Subdivider. Each patch subdivision results in four subpatches. Each subpatch has the same data structure as the parent patch. The subpatches are tested to see if they can be written into the Z-buffer by the Display Processor. Those patches which must be further subdivided are added to the stack.

The Display Processor determines the intensity or grey level and distance for each displayed point and writes them into the Z-buffer. The Z-buffer consists of a frame buffer which stores the pixel intensity values and a depth buffer which contains the distance to each displayed point. When priority conflicts arise, the distances are compared by the Display Processor and the closest point is written into the Z-buffer. The frame buffer is finally scanned in raster fashion to display the picture.

REQUIREMENT ANALYSIS

In order to specify the computations and memory required for each of the functions shown in Figure 15, it was necessary to perform a detailed requirements analysis.

An overview of the approach to this task is shown in Figure 16. The general philosophy is to characterize the computational requirements of each of the algorithm stages analytically as functions of the flight, sensor and terrain. This characterization serves several purposes:

- It helps understand explicitly how each parameter affects the computational requirements of each stage of the algorithm.
- It enables the prediction of the performance requirements for varying flight sensor and terrain parameters; the analysis is not restricted to a given sensor and mission.
- The computer simulation does not have to be run for all possible cases. This is prohibitively expensive besides yielding little insight into the behavior of the system under varying conditions.

The approach has been to divide the requirements analysis problem into two parts--the geometry-dependent part and the algorithm-dependent part. The former is amenable to mathematical analysis and allows direct use of all the flight, sensor, and terrain parameters. The algorithm-dependent part is harder to quantify analytically, but a few runs of the simulation on

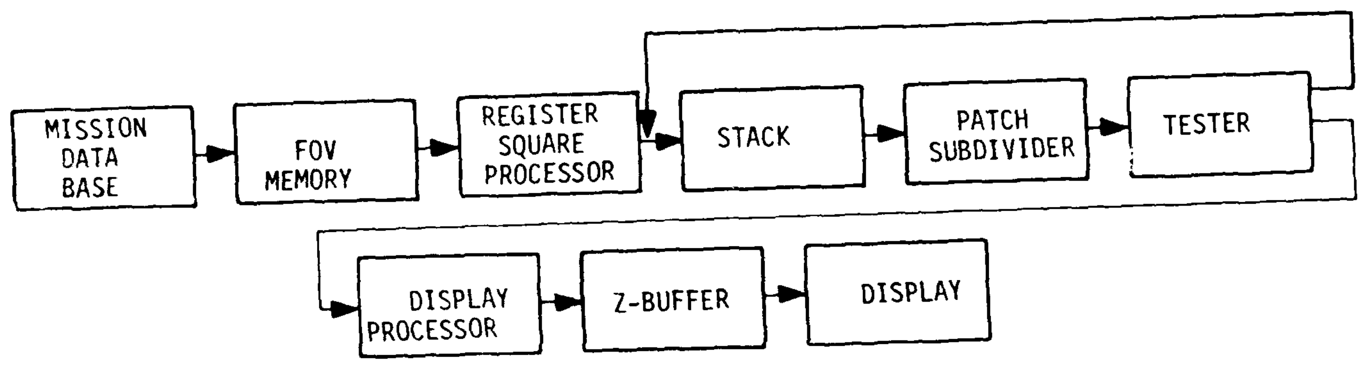


Figure 15. Basic System Functions

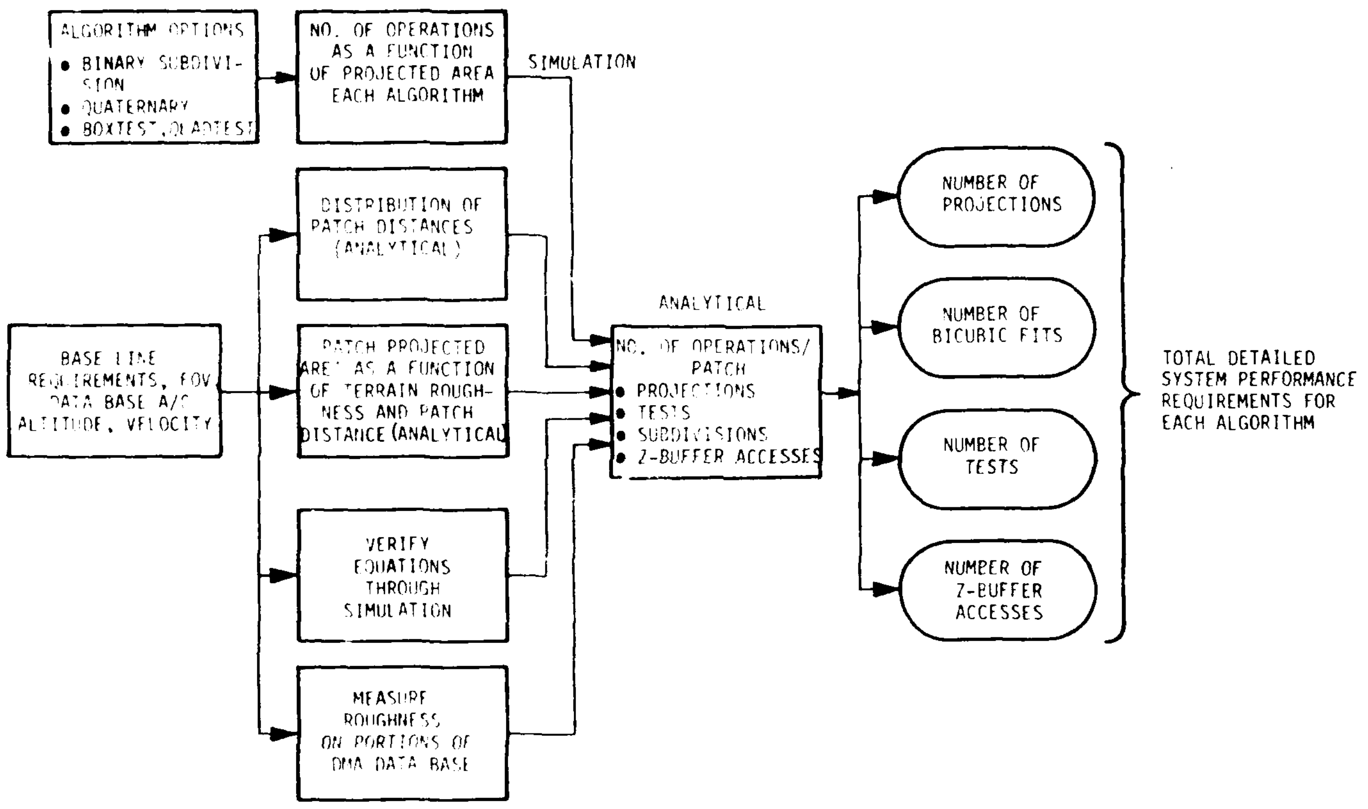

ALGORITHM OPTIONS
• BINARY SUBDIVISION
• QUATERNARY
• BOXTEST, QUADTEST
NO. OF OPERATIONS AS A FUNCTION OF PROJECTED AREA EACH ALGORITHM
SIMULATION
DISTRIBUTION OF PATCH DISTANCES (ANALYTICAL)
BASE LINE REQUIREMENTS, FOV DATA BASE A/C ALTITUDE, VELOCITY
PATCH PROJECTED AREA AS A FUNCTION OF TERRAIN ROUGHNESS AND PATCH DISTANCE (ANALYTICAL)
VERIFY EQUATIONS THROUGH SIMULATION
MEASURE ROUGHNESS ON PORTIONS OF DMA DATA BASE
ANALYTICAL
NO. OF OPERATIONS/ PATCH
• PROJECTIONS
• TESTS
• SUBDIVISIONS
• Z-BUFFER ACCESSES
NUMBER OF PROJECTIONS
NUMBER OF BICUBIC FITS
NUMBER OF TESTS
NUMBER OF Z-BUFFER ACCESSES
TOTAL DETAILED SYSTEM PERFORMANCE REQUIREMENTS FOR EACH ALGORITHM

Figure 16. Overview of Detailed Requirements Specifications Task

a limited test data set have provided statistics on the algorithm performance for each algorithm option. The results of these statistics have been combined with the analytically derived equations for the geometry-dependent part to yield the detailed requirements for each algorithm. These detailed requirements are explicitly parameterized in terms of the flight, sensor, and terrain parameters.

Algorithm Options

Two alternate approaches to the perspective mapping function (Figure 17) were considered. They are projection before subdivision and projection after subdivision. The computational requirements trade-off between the two approaches suggests the former over the latter, as discussed in Section III. The detailed requirements were, therefore, derived for the projection before subdivision approach.

An alternative approach to patch subdivision--adaptive binary subdivision--was devised, as described in Section III. A simpler "box" test was also investigated as a grid coverage test in lieu of the quadrilateral coverage test. These have been incorporated into the simulation software. Three versions of the computer simulations now exist: (a) binary subdivision with quadrilateral test, (b) binary subdivision with box test, and (c) quaternary subdivision with quadrilateral test, the version on STARS.

The processing and computation required to display a patch is a function of the projected area of the patch on the screen. Analytical equations were derived (see Appendix C) for the projected area of a patch as a function of the patch distance, the observer height from the patch and a measure of the terrain "roughness." The analysis indicates that the processing required to display rough terrain can be nontrivially greater than that for flat terrain. The equations form the basis of the requirements analysis and were verified by simulation.

The processing required to display a patch as a function of its projected area on the screen was quantified for each of the three versions of the terrain representation simulation. To this end, the three versions were run with a sequence of patches at varying distances and orientations from the observer. Several statistics were gathered and analyzed through histograms and scatter plots. In particular, the orientation dependence of the patch subdivision process (binary vs. quaternary), the relative efficiency of the box test vs. the quadrilateral test, the number of sub-

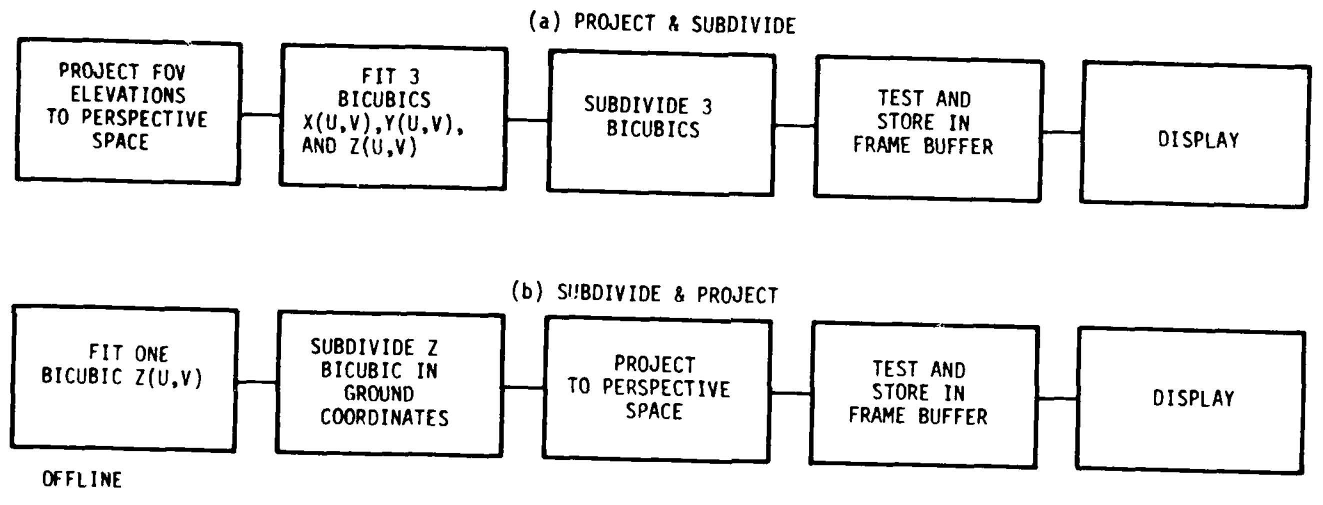

(a) PROJECT & SUBDIVIDE
PROJECT FOV ELEVATIONS TO PERSPECTIVE SPACE
FIT 3 BICUBICS X(U,V),Y(U,V), AND Z(U,V)
SUBDIVIDE 3 BICUBICS
TEST AND STORE IN FRAME BUFFER
DISPLAY
(b) SUBDIVIDE & PROJECT
FIT ONE BICUBIC Z(U,V)
SUBDIVIDE Z BICUBIC IN GROUND COORDINATES
PROJECT TO PERSPECTIVE SPACE
TEST AND STORE IN FRAME BUFFER
DISPLAY
OFFLINE

Figure 17. Two Options for Perspective Rendering

divisions, coverage tests, intensity computations, and frame buffer accesses per patch as a function of the projected area of the patch were quantified.

A terrain roughness model was developed. This model directly applies to the non-linear terrain representation approach. In this model, terrain roughness is quantified by two parameters: (a) the average slope of a terrain patch, and (b) the standard deviation of the elevations of the patch corner points. The roughness parameters have been explicitly incorporated into the projected area equations. A computer program was written to evaluate the terrain roughness statistics from sections of the DMA data base in order to quantify the roughness parameter values for representative terrain types.

An equation was also derived for the distribution patch distances. In combination with the equation for projected areas, the distance equation provides the total projected area as a function of the visual or sensor FOV and the distance to the farthest patch to be mapped. These equations, together with the simulation runs giving the statistics of the algorithm performance, predict the number of times an operation has to be performed at each stage of the process as a function of sensor, flight, and terrain parameters. The critical parameters are:

- Number of patches in the FOV
- Number of patches which need bicubic subdivision
- Number of bicubic subdivisions
- Number of Z-buffer accesses

Number of Patches in FOV

Number of patches in the FOV is given by the equation:

$$P = \frac{\psi_H(\ell_2^2 - \ell_1^2)}{2h^2} \simeq \frac{\psi_H \ell_2^2}{2h^2}$$

where ψ_H is the horizontal FOV (in radians) of the sensor being simulated; ℓ_2 is the distance to the "horizon," the farthest patch being mapped; and h is the patch size on the ground.

Using a nominal $\psi_H = 60^\circ$, $\ell_2 = 15$ miles and $h = 300$ feet, the result is $P = 36,000$ patches in the FOV.

Number of Patches Needing Bicubic Subdivision

The idea here is that not all patches which require subdivision have to be subdivided using the bicubic subdivision. If the projected patch area is small (for example, $<m$, a threshold), then the register square computation can be eliminated, and processing proceeds directly to the patch subdivision. The subdivision on these patches would then be simply linear interpolation. Thus, the expensive bicubic fits and register square computation on these patches are not required. The number of patches requiring bicubic subdivision is given by:

$$B_m = \frac{2}{m\pi} \quad \frac{\theta m}{\psi_H} \quad n^2$$

Here m is the area of a patch in display grid points beyond which it requires bicubic subdivision; θ_m is the terrain roughness parameter (the average patch slope), and n^2 is the total number of raster points.

The patch tilt θ and the patch altitude were histogrammed for two sections of the DMA data base (Nevada). One was a large 100 km x 100 km section and the other was the mountainous 5 km x 5 km section that was used for image generation examples. These histograms are shown in Figures 18a, 18b, 19a, and 19b. It can be seen that the average θ, θ_m, for the mountainous terrain is about 10°; this is the value used in the worst case example for the requirements analysis.

Assuming $\theta_m = 10^\circ$ (rough terrain), $n = 1000$ (high resolution display), and $m = 10$, $B_m \simeq 11,000$. However, for the hardware specification it was assumed that all patches in the 60° FOV ($\sim$40,000) require bicubic subdivision.

Total Number of Patch Subdivision, N_s

The number of patch subdivisions required is proportional to the total projected area, A, of the patches. Each of the three versions of the basic algorithm has a different proportionality constant which was determined by simulation. The three versions are:

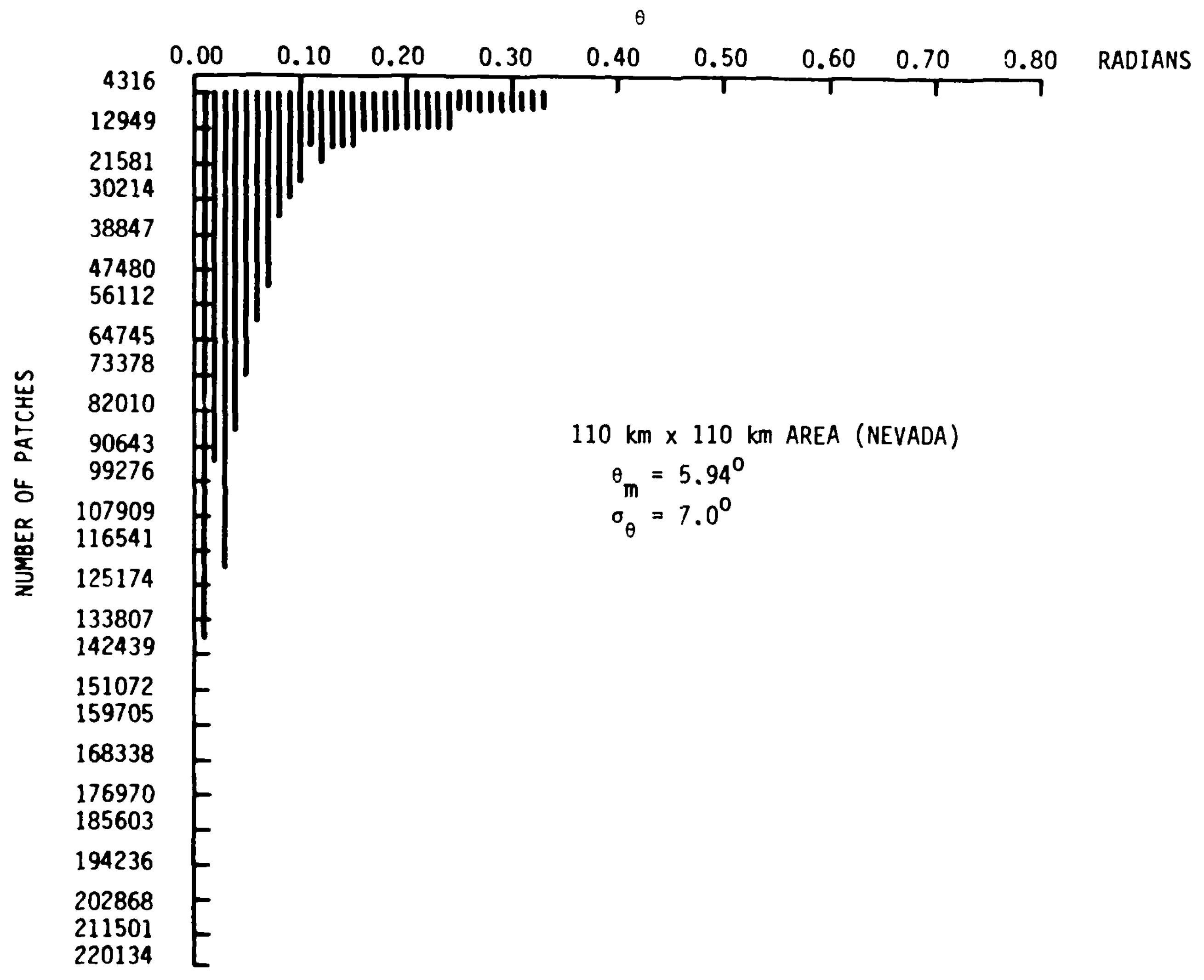


Figure 18a. Histogram of Patch Tilts

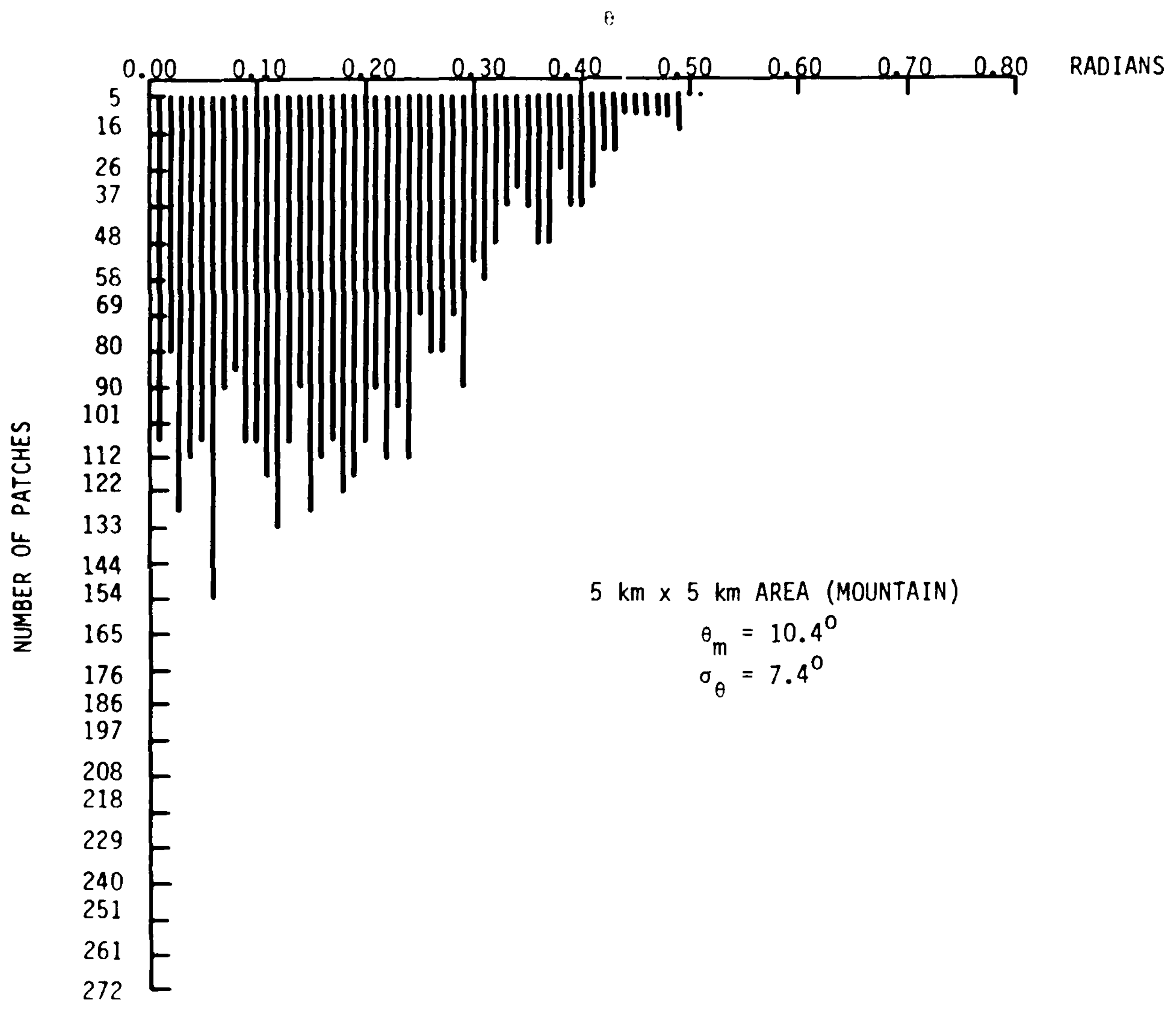


Figure 18b. Histogram of Patch Tilts

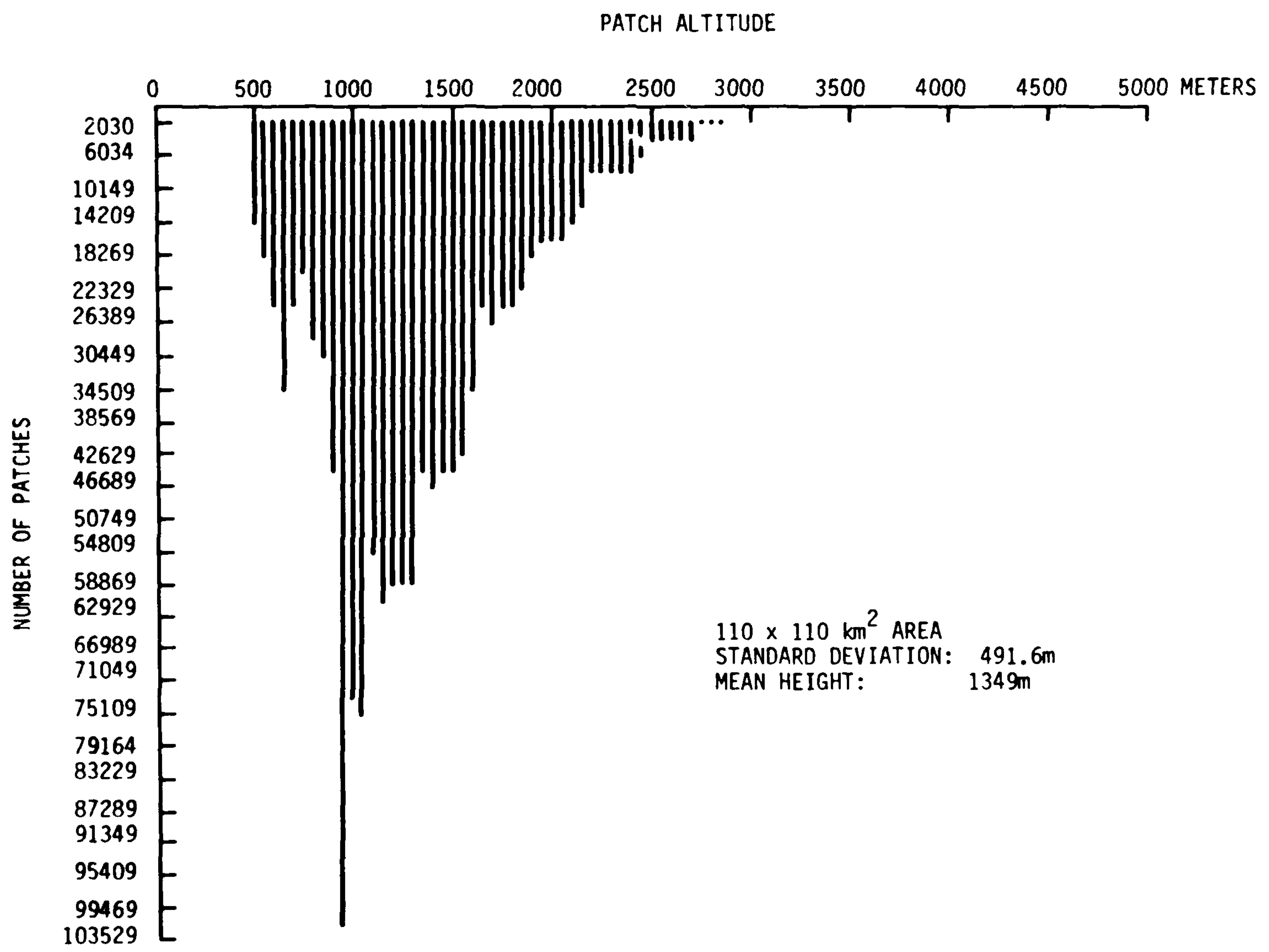


Figure 19a. Histogram of Patch Altitudes

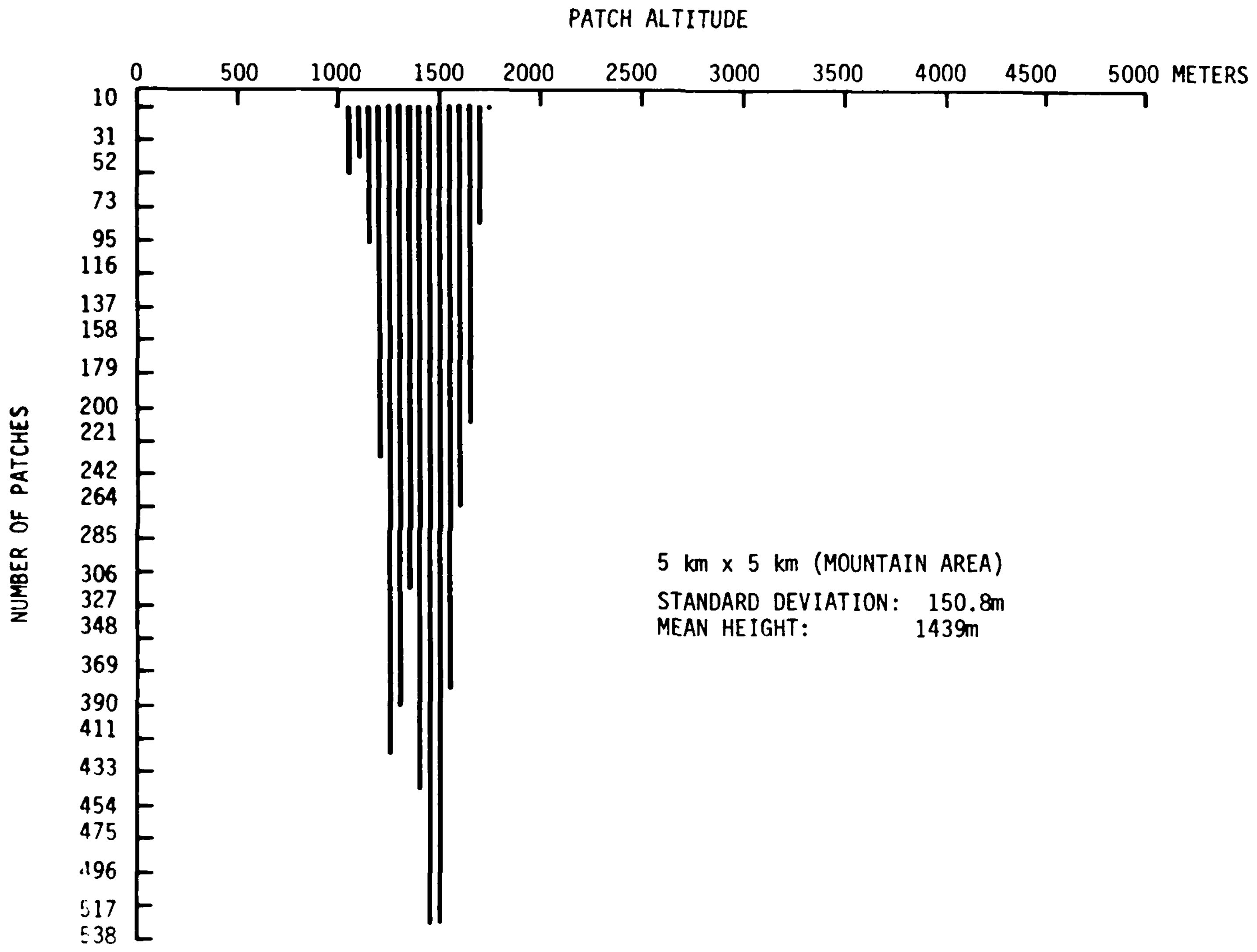


Figure 19b. Histogram of Patch Altitudes

- Binary subdivision with quadrilateral grid test
- Binary subdivision with box test
- Quaternary subdivision with quadrilateral grid test

Each of the three versions was simulated with a set of patches at different distances. The number of patch subdivisions and Z-buffer accesses and the projected patch area were gathered for each patch. These statistics were plotted in scatter plots and analyzed for each algorithm version and patch orientation with respect to the viewpoint (Figures 20, 21, and 22). From these the constants of proportionality that link the total number of subdivisions and Z-buffer accesses for each algorithm to the total projected area (which is an explicit function of the sensor flight and terrain parameters) were determined. It is observed that the box test for grid coverage appears to require many more subdivisions than other versions (using quadrilateral test for the 45° patch orientation case).

Assuming that all patch orientations are equally likely, the results were:

$$N_s = \begin{cases} 1.5A \text{ (Quaternary)} \\ 1.3A \text{ (Binary with quadrilateral test)} \\ 2.2A \text{ (Binary with box test)} \end{cases}$$

Note that these are equivalent <u>two-way subdivisions</u> (one quaternary subdivision is computationally equivalent to two binary subdivisions). The total projected area is derived in Appendix C and is given by:

$$A = \zeta \psi_H \left[\frac{\ell_2 - \ell_1}{\ell_1 \ell_2} \right] + \frac{2}{\pi} \theta_m \psi_H \ln \left(\frac{\ell_2}{\ell_1} \right) \quad \text{rad}^2$$

where ζ is the height of the viewpoint. Under simplifying approximations, this becomes (see Eq. C-15, Appendix C):

$$A = n^2 \left[1 + \frac{12}{\pi} \frac{\theta m}{\psi_v} \right]$$

raster points, which explicitly shows the effect of θ_m, the terrain roughness parameter.

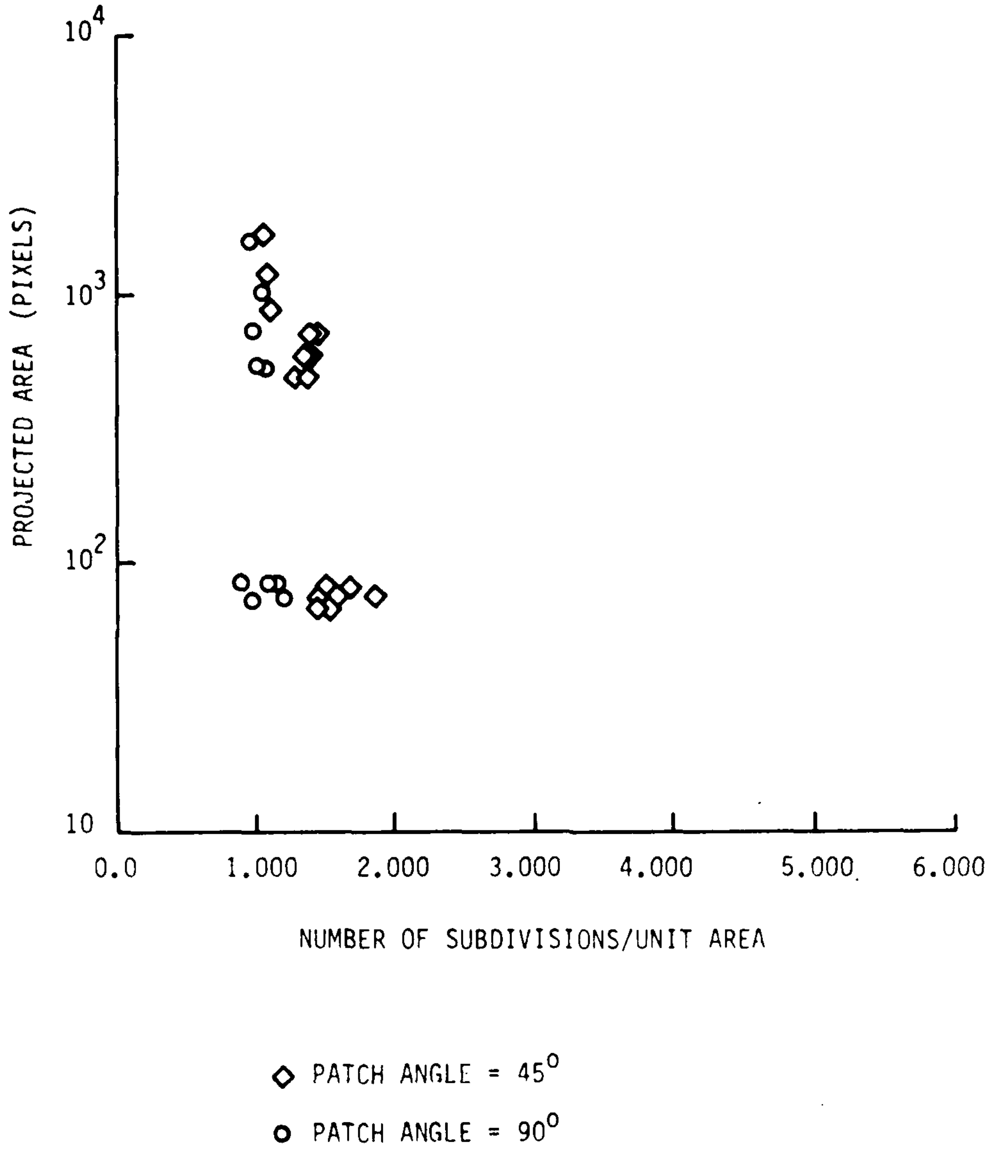


Figure 20. Scatter Plot of Projected Area--Binary (Quadrilateral Test)

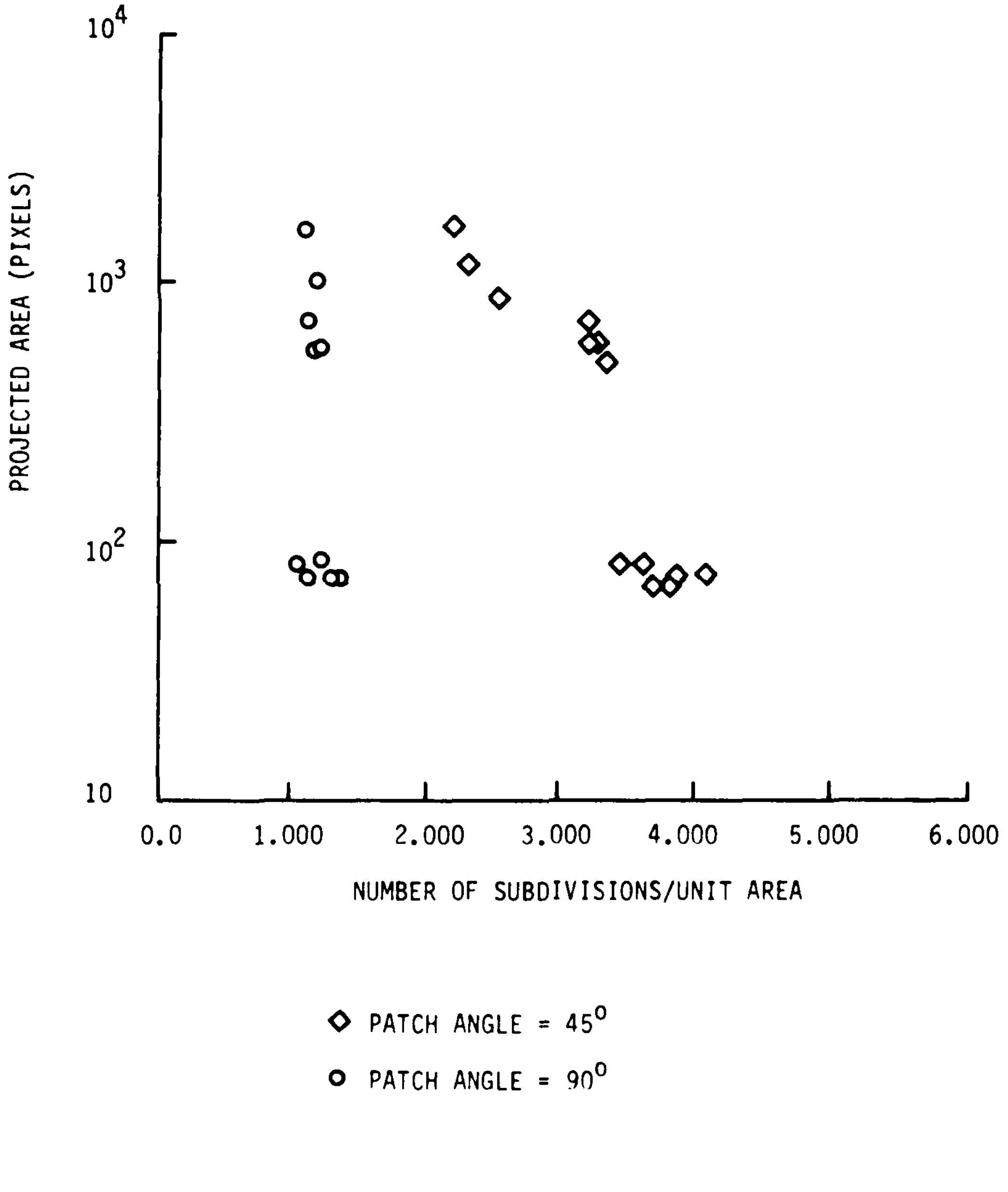
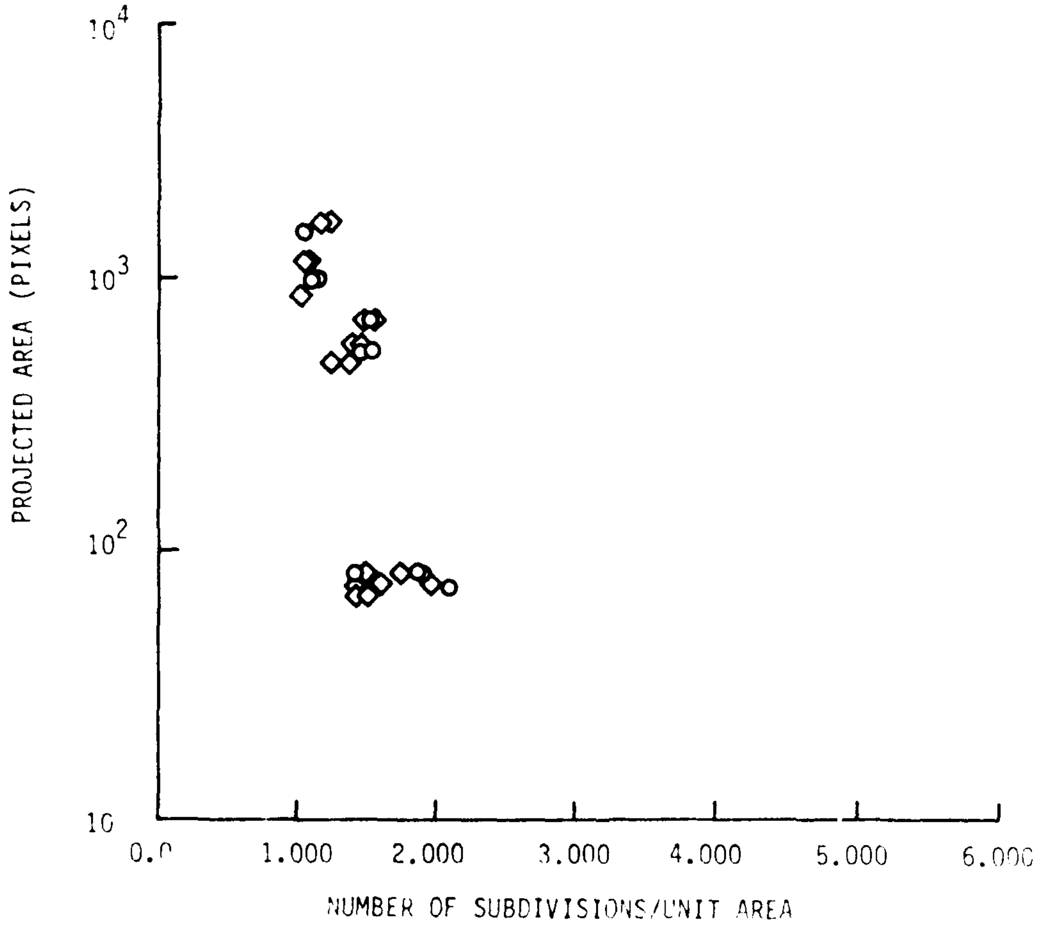


Figure 21. Scatter Plot of Projected Area--Binary (Box Test)



◇ PATCH ANGLE = 45^0

○ PATCH ANGLE = 90^0

Figure 22. Scatter Plot of Projected Area--Quaternary (Quadrilateral Test)

Assuming $\theta_m = 15^\circ$ for a worst case terrain roughness, a vertical FOV $\psi_v = 30^\circ$ gives a projected area:

$$A = n^2 \ [2.27]$$

which implies that a rough terrain, on the average, can have more than two hidden surfaces which map to the same point on the screen. This result and conclusion are discussed in detail in Appendix C.

As a baseline, the quaternary subdivision with the quadrilateral test was chosen for further hardware specification. Although binary subdivision required fewer subdivisions, the resulting data structure would be more complicated. The number of quaternary subdivisions for the given parameters is then given by:

$$N_s = 1.5 \times (1/2)^* \times 2.27\, n^2$$

or

$$N_s \simeq 1.7 \times 10^6/\text{frame}$$

for a display resolution of 10^6 raster points.

Figures 23a, 23b, 23c, and 23d show the predicted projected area vs. distance curves for various heights and average patch tilts θ_m. Figures 24a, 24b, and 24c show the ratio of the average predicted area to the average measured area from the simulation for each patch distance, height, and tilt θ_m. Note that the predicted results track the measured results closely. This is important because the requirements analysis is founded on the projected area equations.

Number of Z-Buffer Accesses

The number of Z-buffer accesses is also a function of the total projected area. In fact, for both versions of the patch subdivision algorithm with the exception of the box test, this constant is unity (that is, one Z-buffer access/unit projected area). This has been verified by computer simulation. Hence,

*The factor 1/2 arises because this is the number of four-way subdivisions.

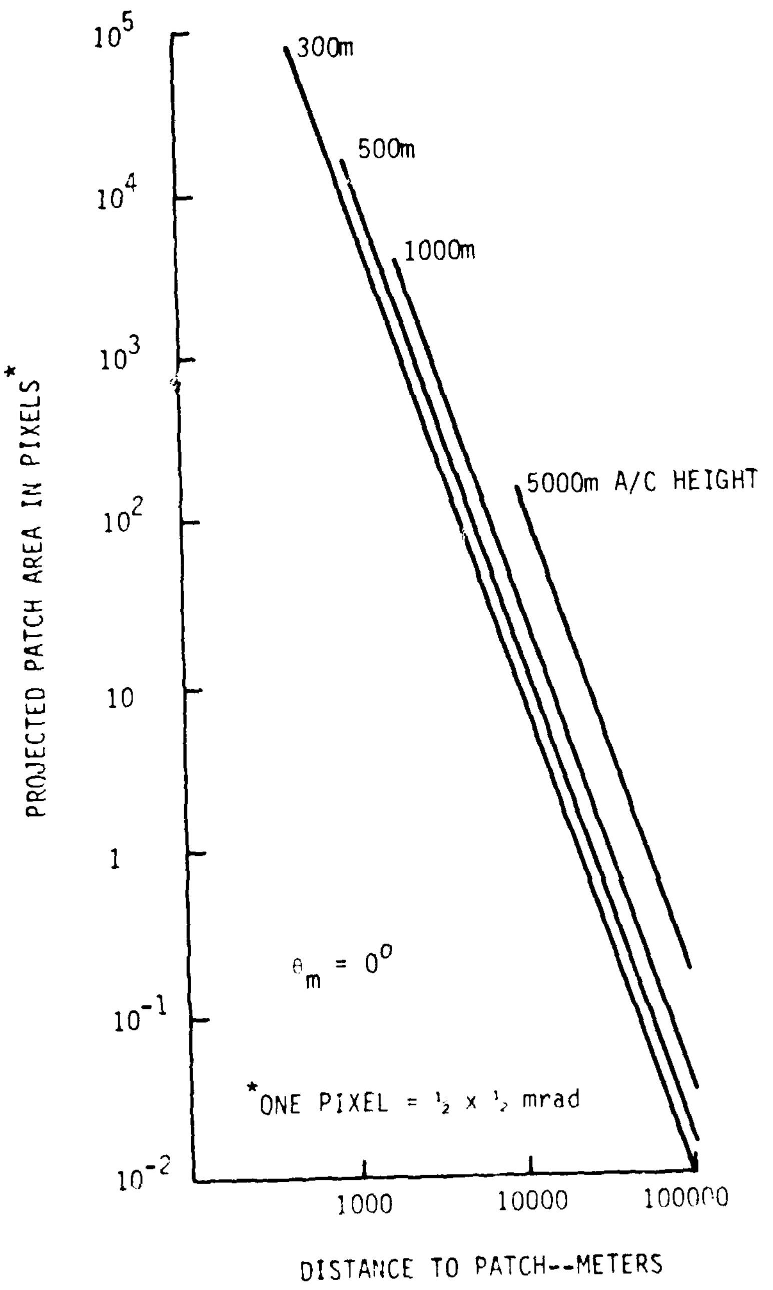


Figure 23a. Projected Area vs. Distance--$\theta_m = 0°$

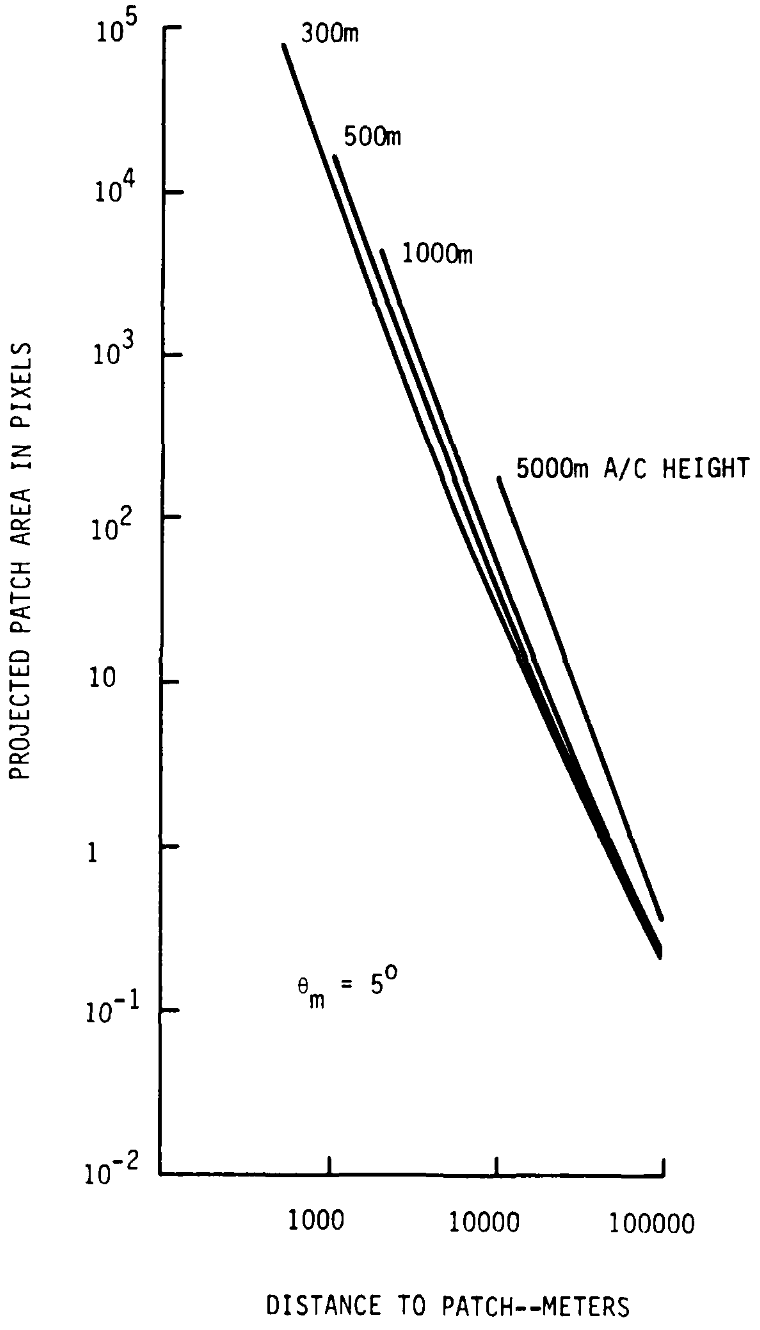


Figure 23b. Projected Area vs. Distance--θ_m 5°

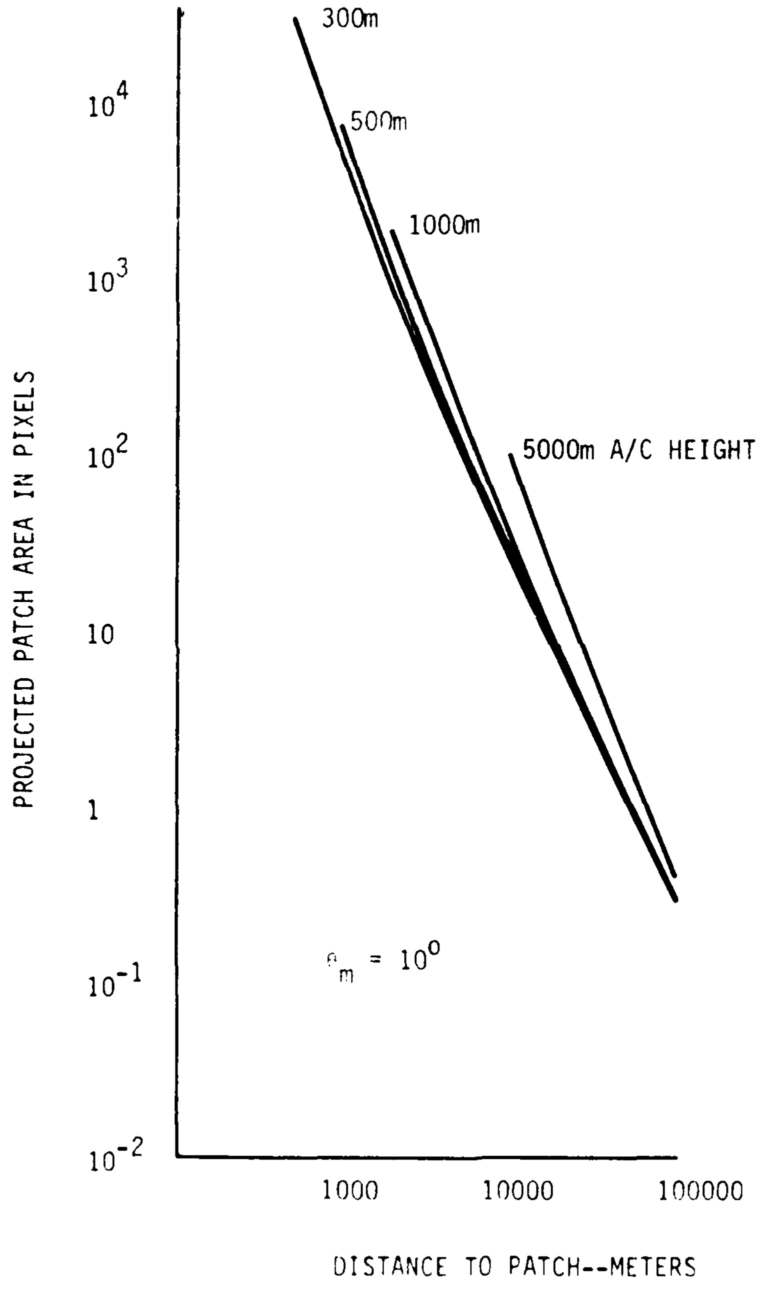


Figure 23c. Projected Area vs. Distance--θ_m = 10

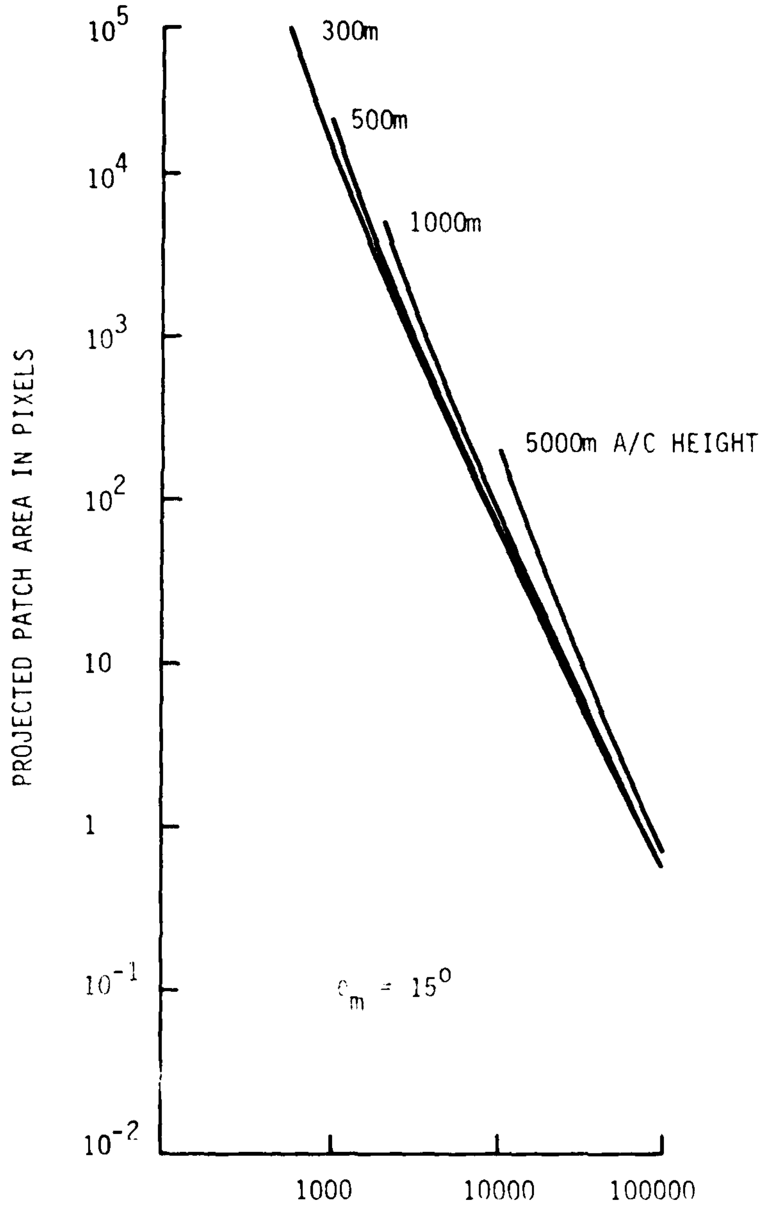


Figure 23d. Projected Area vs. Distance--$\theta_m = 15°$

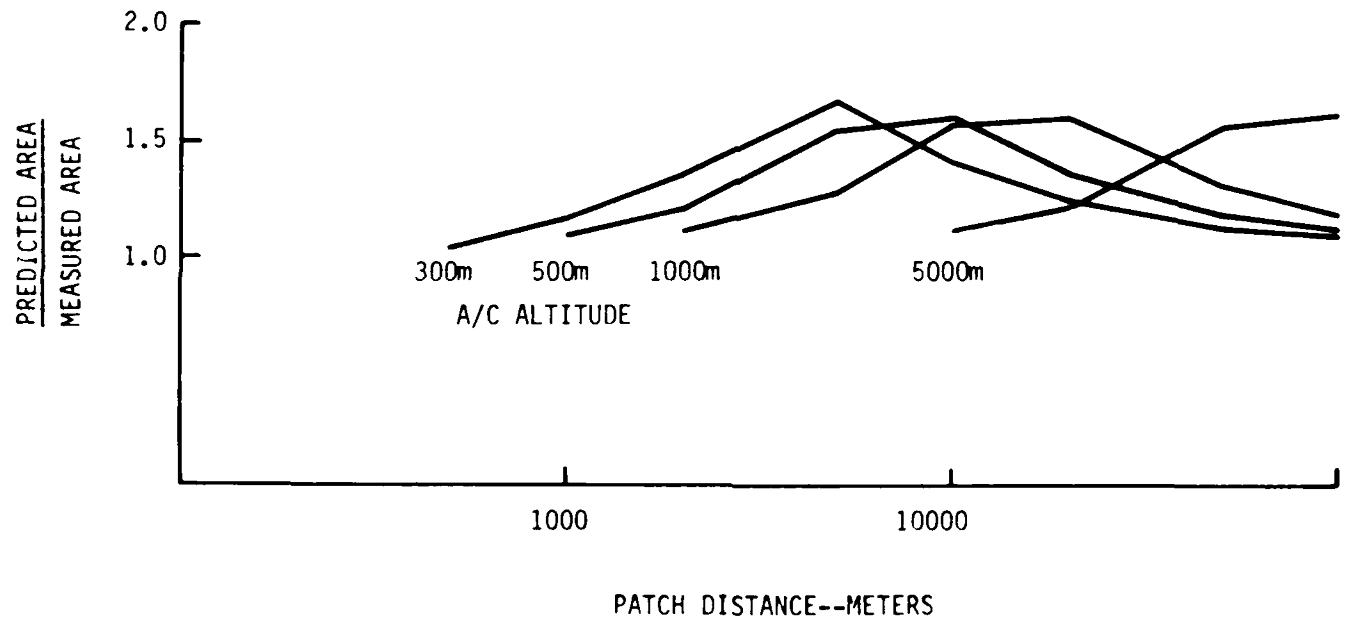


Figure 24a. Comparison of Predicted and Measured Projected Areas--$\theta_m = 5°$

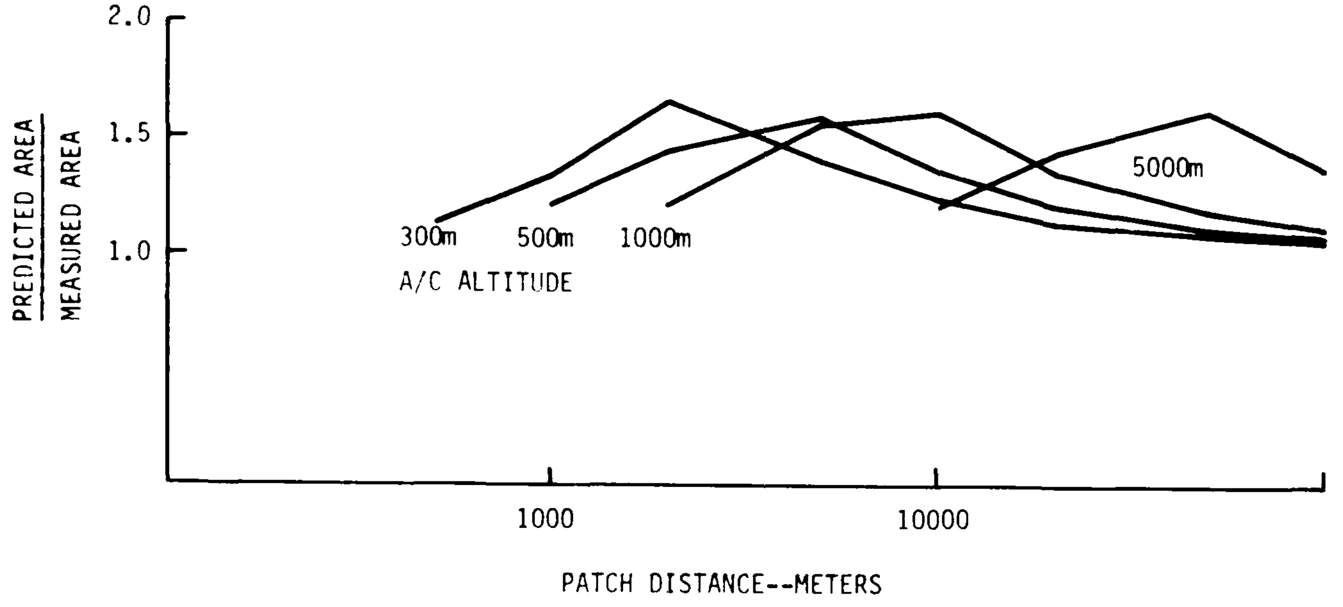


Figure 24b. Comparison of Predicted and Measured Projected Areas--$\theta_m = 10°$

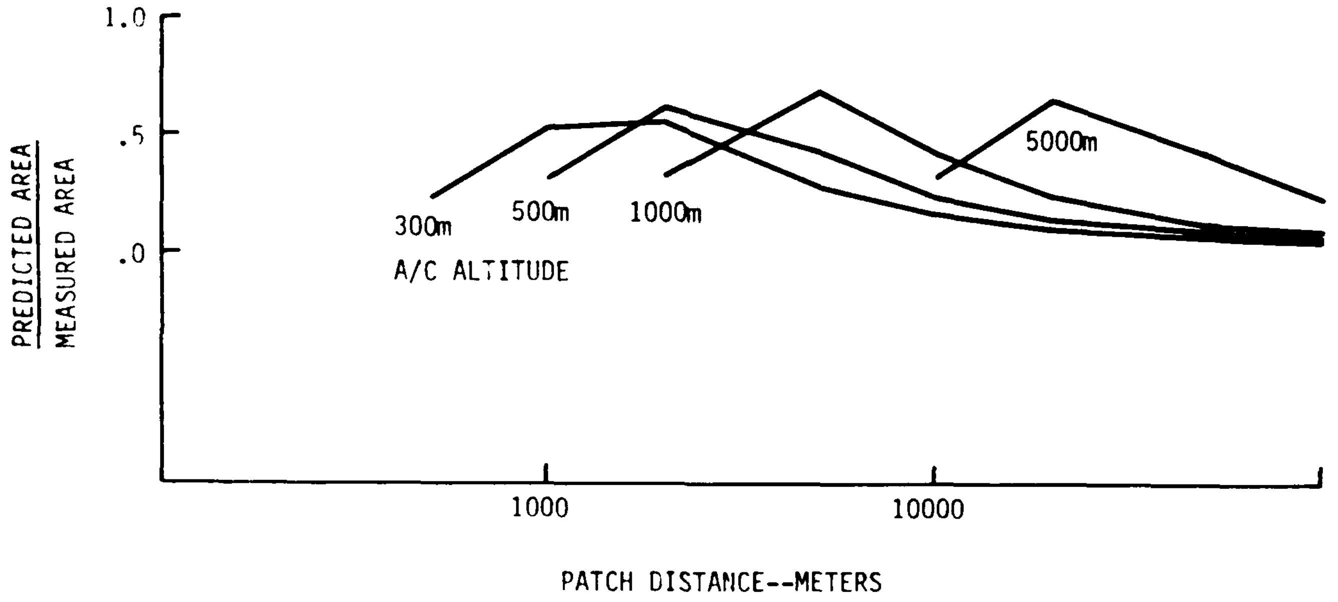


Figure 24c. Comparison of Predicted and Measured Projected Areas--$\theta_m = 15°$

$$NZ = \begin{cases} A \text{ for all algorithm versions, except,} \\ 1.72A \text{ for box test } (\diamond \text{ patch orientation}) \end{cases}$$

For the rough terrain parameter assumed,

$$NZ \simeq 2.27 \times 10^6 .$$

REAL-TIME IMPLEMENTATION REQUIREMENTS

Figure 25 shows the storage and throughput requirements for the various functional units of the system, based on the analysis described in the preceding section. It should be noted that these requirements are conservative, based on worst case requirements in most instances.

Implementation requirements were specified for each functional block of Figure 25. The approach taken was to assume fully parallel computation and to use pipeline architectures where possible for the individual blocks and from block-to-block. In a fully pipelined system, the throughput is determined by the slowest part of the pipeline.

Field-of-View (FOV) Memory

The FOV Memory provides temporary storage for terrain elevation samples for all potentially visible patches. All patches for the mission are stored in a mass memory such as a magnetic disc. However, access to mass memories is too slow for the pipelined Register Square Processor, so the FOV Memory is necessary. The contents of the FOV Memory are continually updated as the aircraft flies across the terrain. A maximum of about 500 patches must be added and subtracted for each new frame. Approximately one terrain elevation sample of 16 bits is required for each patch.

The FOV memory is organized like a toroid in that addresses which exceed the physical size of the memory wrap around and begin again at the initial addresses. This organization allows for rapid scanning of all patches within the FOV and easy updating with new patches as the aircraft moves across the terrain.

Once every frame, the terrain elevation data are scanned out of the FOV Memory and processed by the Register Square Processor.

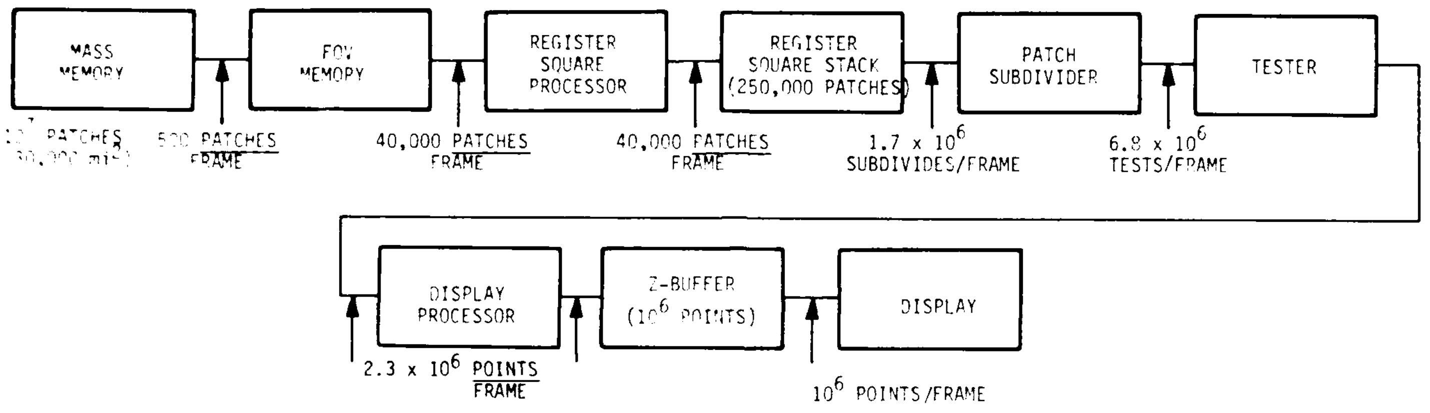


Figure 25. Basic System Functions--Requirements

Register Square Processor

The Register Square Processor subsystem is a pipelined processor which converts all relevant terrain patch data into bicubic form in image screen coordinates for subdivision and display.

The separate functional units of the Register Square Pipeline Processor are shown in Figure 26. The first block (Stage 1) transforms the terrain elevation samples to perspective space. The required calculations are shown in Figure 27. A pipelined hardware implementation of this function is shown in Figure 28.

The pipeline contains nine multipliers, nine adders, four registers, and two dividers and requires five clock cycles for execution. The throughput is determined by the slowest operation, the divide. Assuming a conservative 16-bit divide time of 200 nsec (typical current generation off-the-shelf device speeds), all 40,000 terrain samples in a 60° FOV can be projected in 8×10^{-3} sec. Inputs to the projection hardware, in addition to the terrain samples, are aircraft position coordinates, heading, roll, and pitch. The outputs are the terrain elevation samples expressed in image screen coordinates.

To determine the bicubic patch coefficients, it is necessary to collect the terrain elevation samples in overlapping groups of 16 samples/patch. This is the function of the next stage, called the circular buffer (Stage 2). For each patch, the 16 sample vectors defining the patch are used in Stage 3 to determine the bicubic fit coefficients, which in turn are used in Stage 4 to compute the "register squares." The register squares are the quantities used in the patch subdivision.

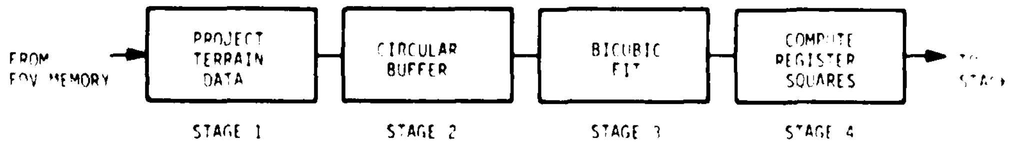


Figure 26. Functional Elements of Register Square Pipeline Processor

PRECISION $X,Y,Z \sim 10$ BITS

$X_o, Y_o, Z_o \sim 16$ BITS $\quad A \sim 16$ BITS

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} X-X_o \\ Y-Y_o \\ Z-Z_o \end{bmatrix}$$

$$\begin{bmatrix} X_e \\ Y_e \\ Z_e \end{bmatrix} = \begin{bmatrix} X'/Z' \\ Y'/Z' \\ Z' \end{bmatrix}$$

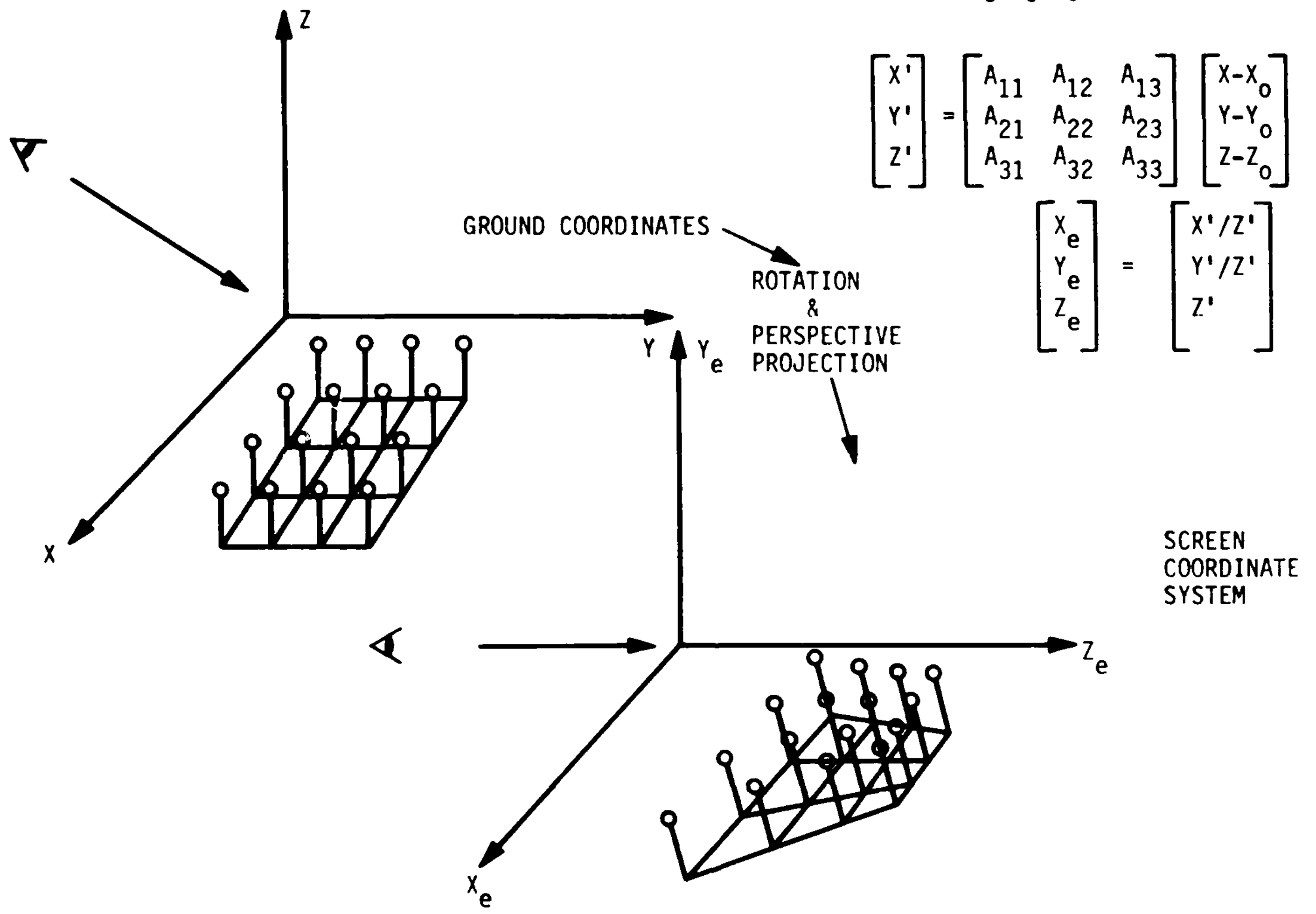
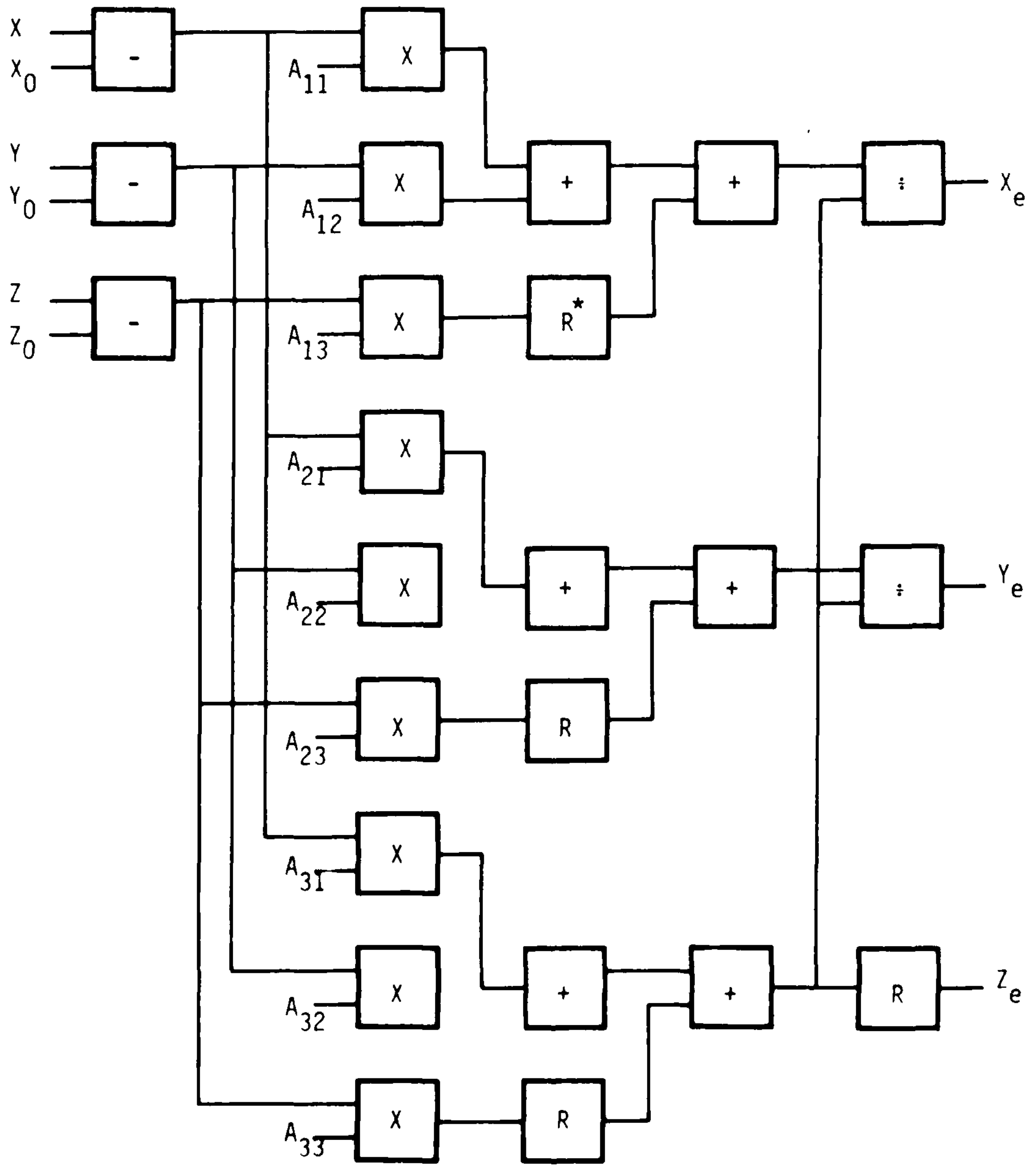


Figure 27. Perspective Projection of Elevations



* R DENOTES REGISTER

Figure 28. Pipeline Hardware Block Diagram for Perspective Projection Stage

The computations required for the bicubic fit coefficients are described in Appendix A and are summarized in Figure 29. This stage requires the multiplication of the 4 x 4 array of 16 projected terrain sample vectors surrounding each patch by a constant 4 x 4 matrix M (premultiplication) and its transpose (post multiplication).

A pipelined hardware approach to the bicubic fit (Stage 3) is shown in Figure 30. Each x, y, and z component requires 32 identical modules, each consisting of four multipliers and three adders. Sixteen registers are also required for the 4 x 4 constant matrix M. Six clock periods are required for completion of each patch. Again, the throughput rate is conservatively estimated at 5 x 10^6 patches/second.

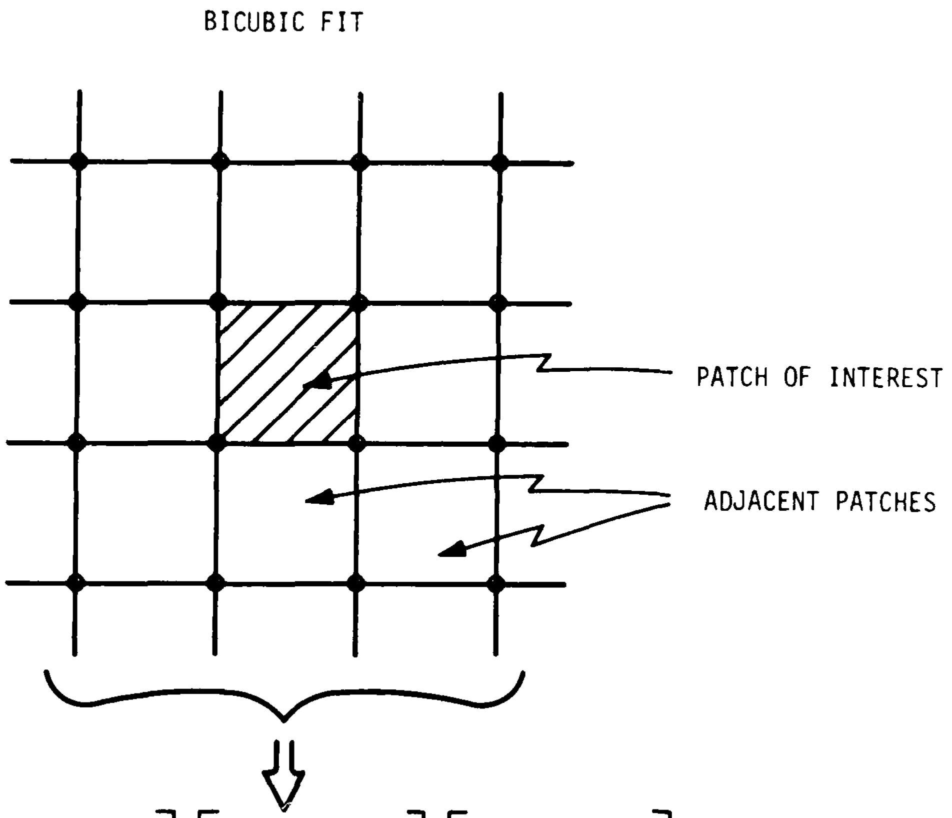
The output of the bicubic fit stage consists of a 4 x 4 matrix A. There is an A matrix for each of the three components. The elements of the A matrix are denoted a_{ij}.

The next step (Stage 4) consists of computing the 16 register square entries from the a_{ij} for each of the three components. The definitions of the register square entries are shown in Figure 31 and are derived in Appendix B. A pipelined implementation of this stage is shown in Figure 32. A total of 27 adders and 49 registers are required for parallel implementation. Each patch requires four clock periods.

In summary, the Register Square Processor performs the functions of perspective projection and the fitting of the bicubic polynomial to the terrain samples. The output is in the form of three arrays of 16 numbers which are used directly by the succeeding bicubic subdivision algorithm. This processor is fully pipelined and can process an FOV of 40,000 patches in 8 msec. The hardware required consists of 393 multipliers, two dividers, 327 adders, and 72 registers.

Register Square Stack

The outputs from the Register Square Processor, as well as the previously subdivided patches which require further subdivision, are stored in a memory called the Register Square Stack until they can be processed by the Patch Subdivider. This memory can be organized as either a first in-first out (FIFO) or last in-first out (LIFO) stack.



$$\underset{(4 \times 4)}{\underset{M}{\begin{bmatrix} m_{11} & \cdot & \cdot & m_{14} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ m_{41} & \cdot & \cdot & m_{44} \end{bmatrix}}} \underset{(4 \times 4)}{\underset{Z}{\begin{bmatrix} z_{11} & \cdot & \cdot & z_{14} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ z_{41} & \cdot & \cdot & z_{44} \end{bmatrix}}} \underset{(4 \times 4)}{\underset{M_T}{\begin{bmatrix} m_{11} & \cdot & \cdot & m_{41} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ m_{14} & \cdot & \cdot & m_{44} \end{bmatrix}}} = \underset{(4 \times 4)}{\begin{bmatrix} A_z \end{bmatrix}}$$

Figure 29. Bicubic Fit

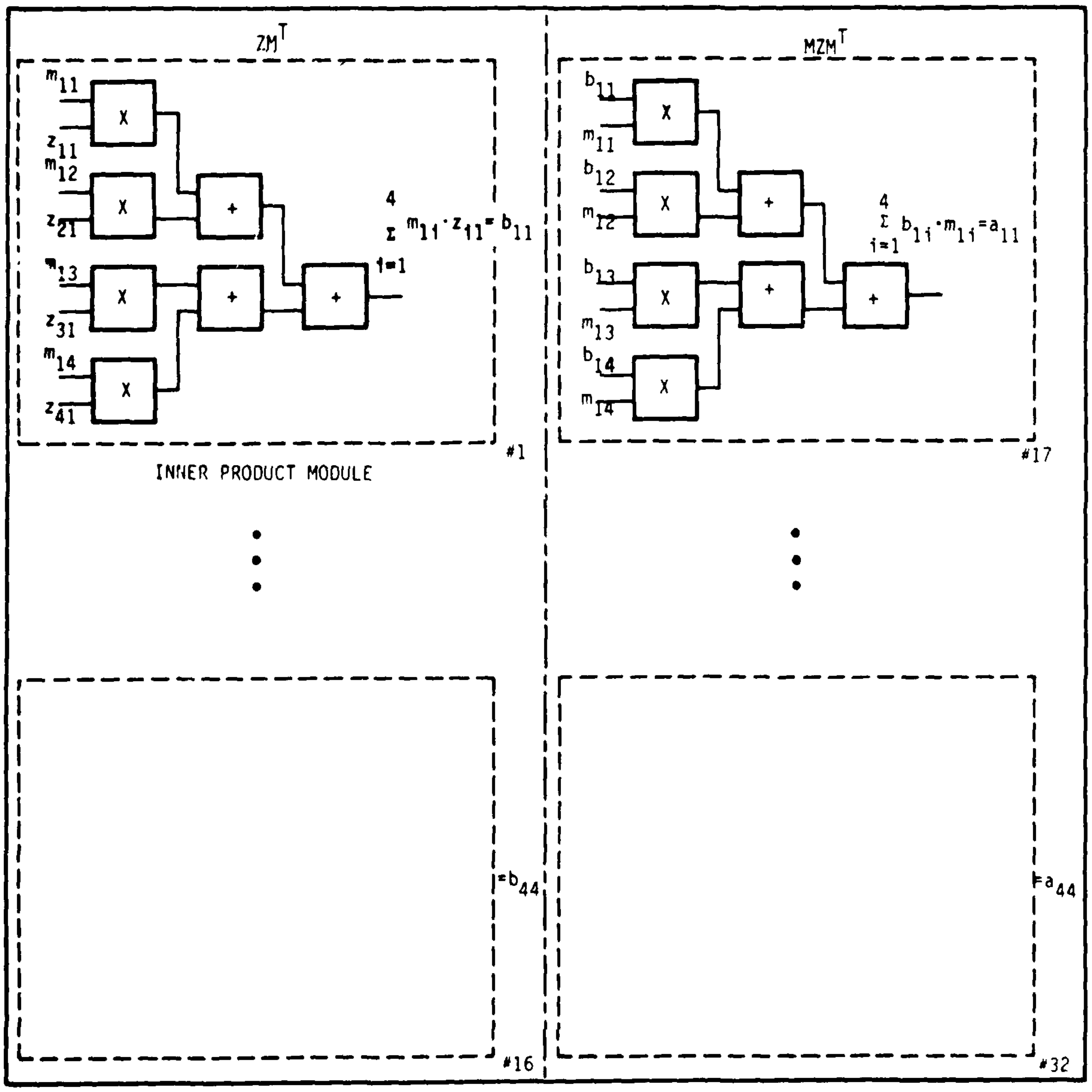


Figure 30. Pipelined Bicubic Fit Block Diagram

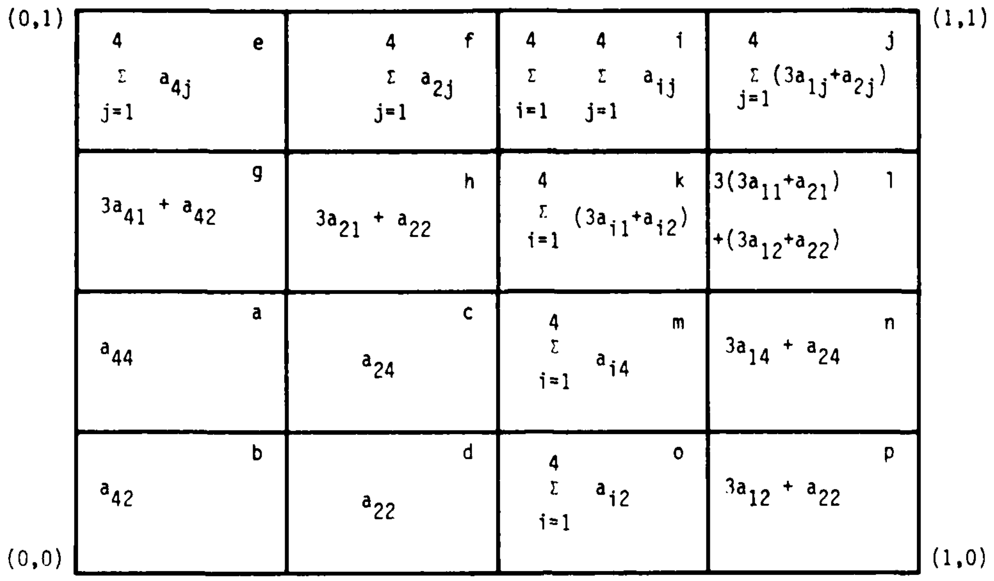

(0,1)
(1,1)
(0,0)
(1,0)
$\sum_{j=1}^{4} a_{4j}$ e
$\sum_{j=1}^{4} a_{2j}$ f
$\sum_{i=1}^{4} \sum_{j=1}^{4} a_{ij}$ i
$\sum_{j=1}^{4} (3a_{1j}+a_{2j})$ j
$3a_{41} + a_{42}$ g
$3a_{21} + a_{22}$ h
$\sum_{i=1}^{4} (3a_{i1}+a_{i2})$ k
$3(3a_{11}+a_{21}) + (3a_{12}+a_{22})$ l
a_{44} a
a_{24} c
$\sum_{i=1}^{4} a_{i4}$ m
$3a_{14} + a_{24}$ n
a_{42} b
a_{22} d
$\sum_{i=1}^{4} a_{i2}$ o
$3a_{12} + a_{22}$ p

Figure 31. Register Squares Prior to Subdivision

It is difficult at this time to predict the size of the stack needed. It will be a function of the input rate of the patches, but, more importantly, it will be the rate at which subdivided patches are returned for further subdivision.

The total memory capacity required is given by

$$\text{Total} = \frac{16}{\text{(Register Squares)}} \times \frac{16}{\text{(Bits)}} \times \frac{3}{(x, y, z)} \times \begin{array}{l}\text{number of patches in the}\\ \text{stack at a given time}\end{array}$$

In the worst case, assume that one patch fills the FOV. The worst case enumerates the subdivision tree breadth first. Since the 10^6 terminal patches do not go back to the stack, all nodes of the quaternary tree have to be stored. Thus, $4/3 \times 10^6/4 = 0.33 \times 10^6$ subpatches may be in the stack. Hence, the maximum size is $0.33 \times 10^6 \times 768 = 256$ megabits. Using 64K bit random access memory (RAM) chips devices, this memory requires 4000 integrated circuits. These can be arranged into blocks to reduce the input-output (I/O) speed requirement of each stage. Note that this is the upper limit. It is quite possible that the memory requirement can be substantially less than this. This can be verified by Monte Carlo simulations.

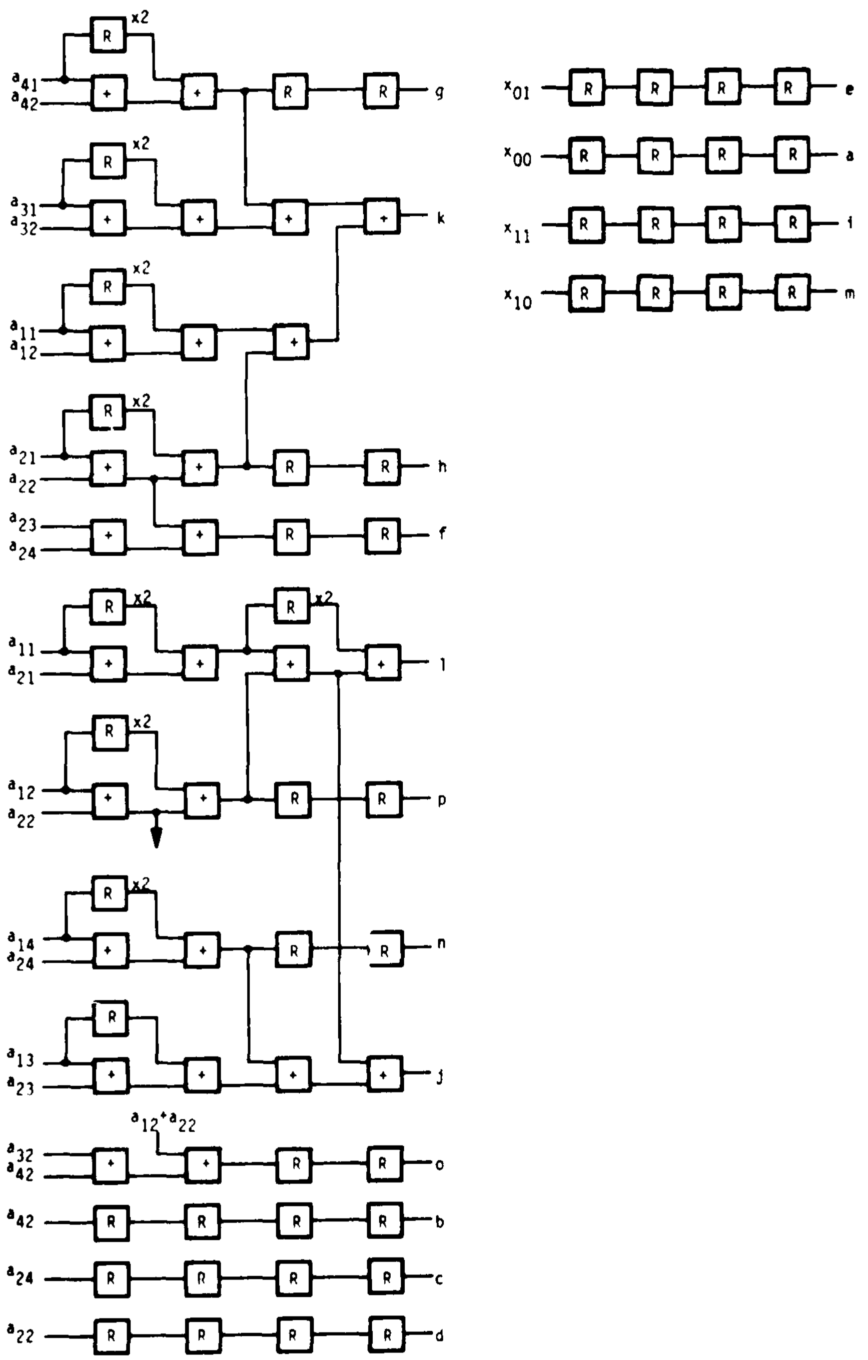


Figure 32. Register Squares Pipeline

PATC[illegible] [illegible]IVIDER

The algor[illegible] [illegible]r [illegible]tch subdivision is described in Appendix B and is shown in Fig[illegible] [illegible]. [illegible] hat subdivision produces nine 2 x 2 register squares from t[illegible] [illegible] 2 x [illegible] iste[illegible] squares of the parent patch. The nine new register squares [illegible] uped to provide four new sets of register squares correspo[illegible] ding to the [illegible] subpatches.

Figures 34a and 34b show the parallel hardware implementation of the subdivide algorithm. It requires 93 adders and 318 registers. With this degree of parallelism, four clock periods are required for subdivision of a bicubic patch x(u, v), y(u, v), z(u, v). For a clock period of 100 nsecs, three patch subdivision pipelines are required.

Tester

After subdivision, each subpatch is tested to see whether it should be further subdivided or passed on to the Display Processor. The present test criterion is that the patch in question should surround one, and only one, display gridpoint. Patches which surround more than one gridpoint are returned for further subdivision.

Two alternatives to approximating the projected patch for the grid test have been considered: the quadrilateral and box approximations (Figure 13).

The box approximation, as can be seen, tends to overestimate the number of grid points surrounded by the patch but results in a simpler implementation. Figures 35 and 36 show the implementation of the box grid test.

For maximum throughput, four grid testers are required for each Patch Subdivider, for a total of 12.

Display Processor and Z-Buffer

The Display Processor is a pipelined processor that determines the tone or intensity for each display subpatch and assembles the picture. In the picture assembly process, any priority conflicts are resolved point-by-point using the Z-buffer.

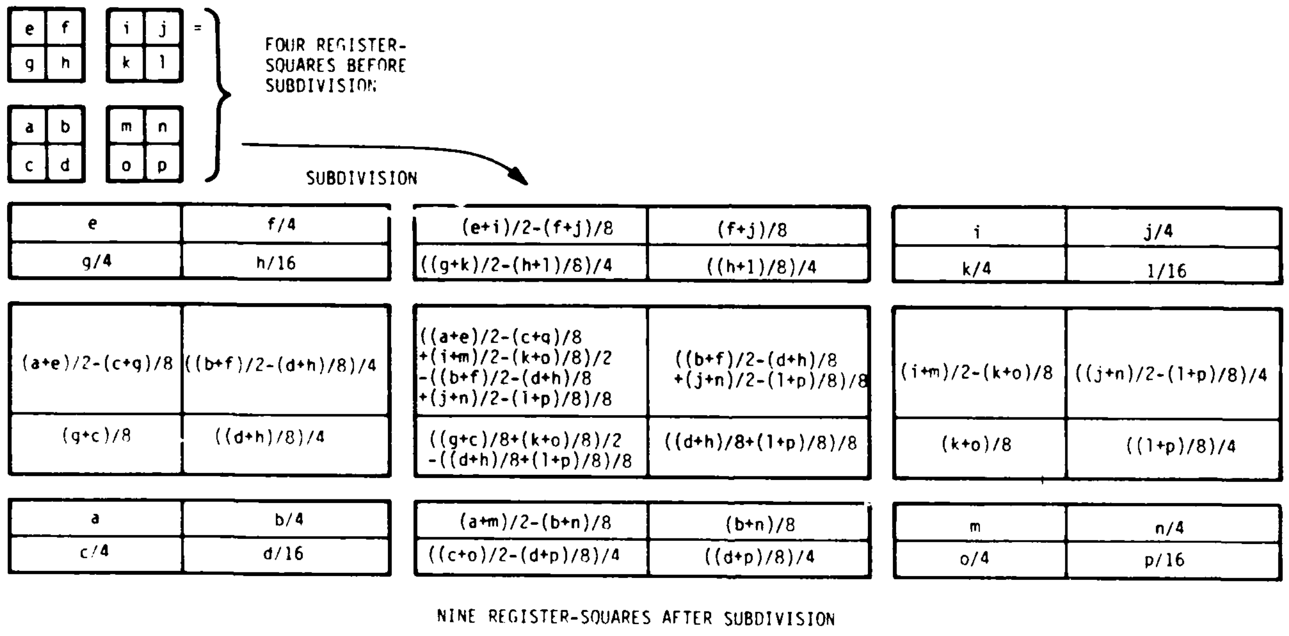
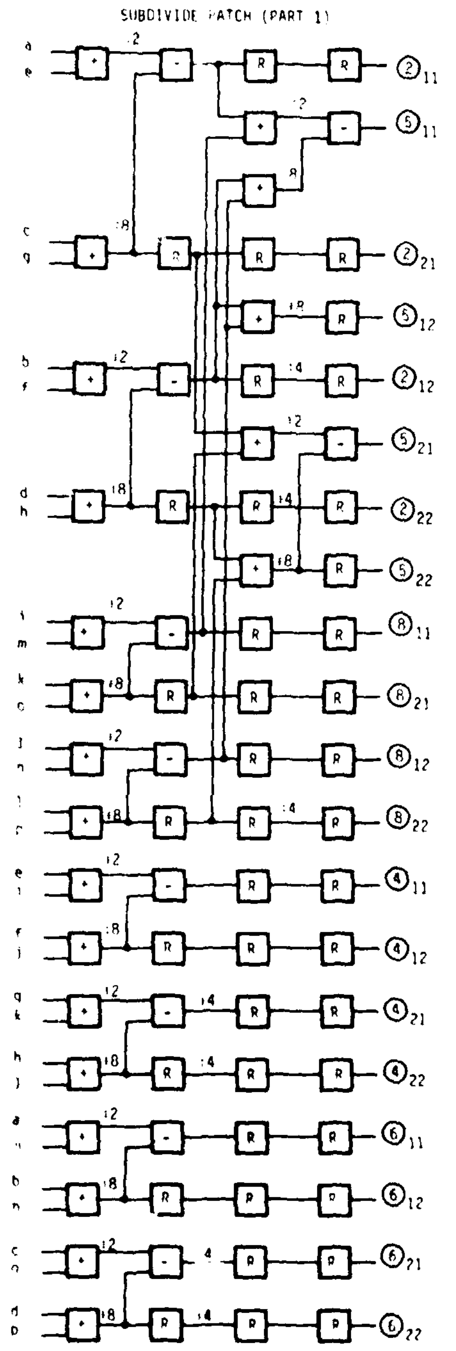


Figure 33. Register Subdivision Equations



NOTE: SUBSCRIPT IDENTIFIES ONE OF FOUR NEW REGISTER SQUARE VALUES IN ONE OF NINE NEW REGISTER SQUARES, DENOTED BY CIRCLED NUMBER.

Figure 34a. Subdivide Pipeline Part I

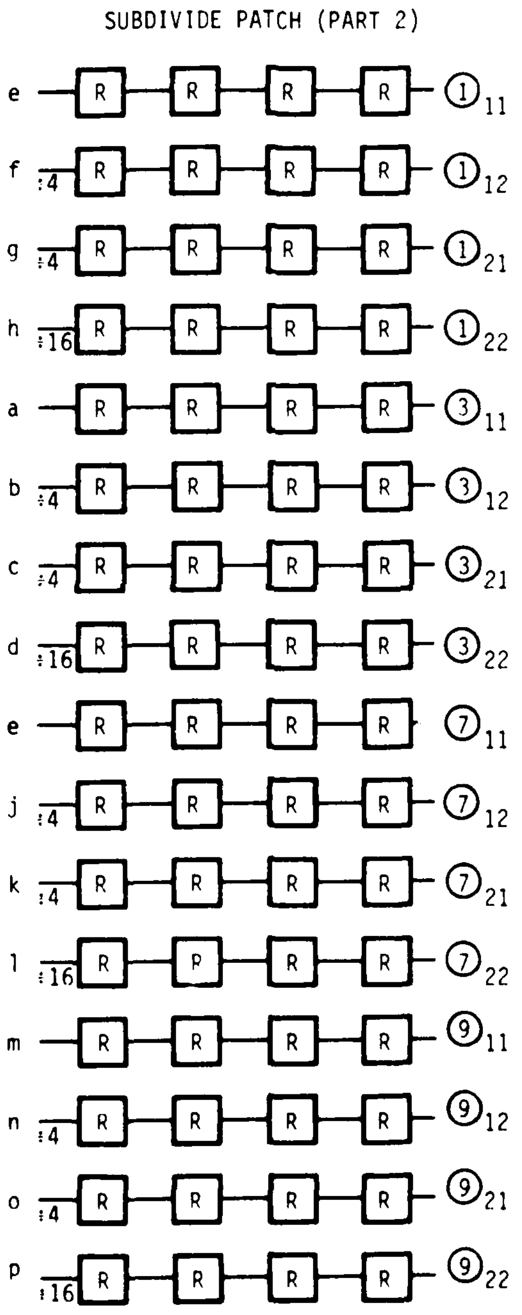
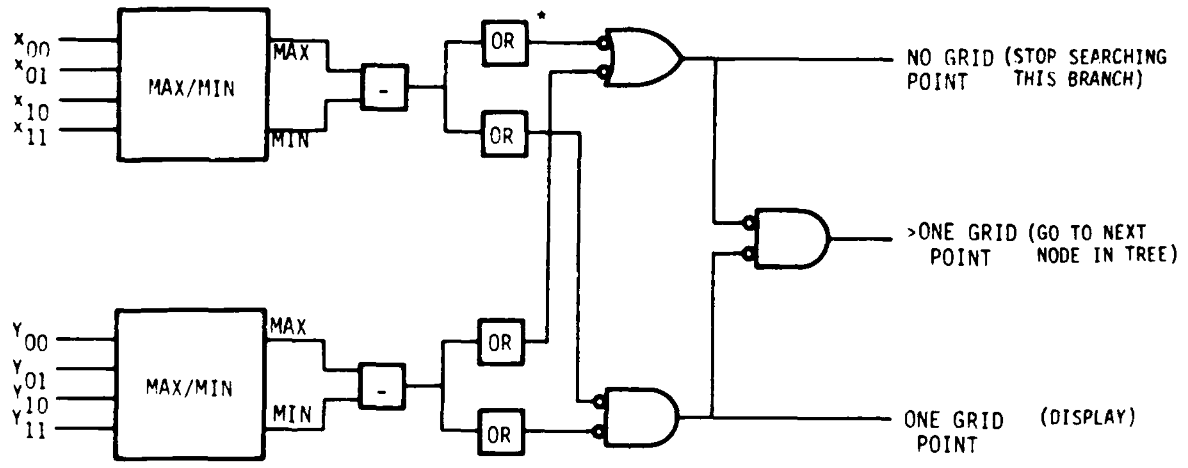
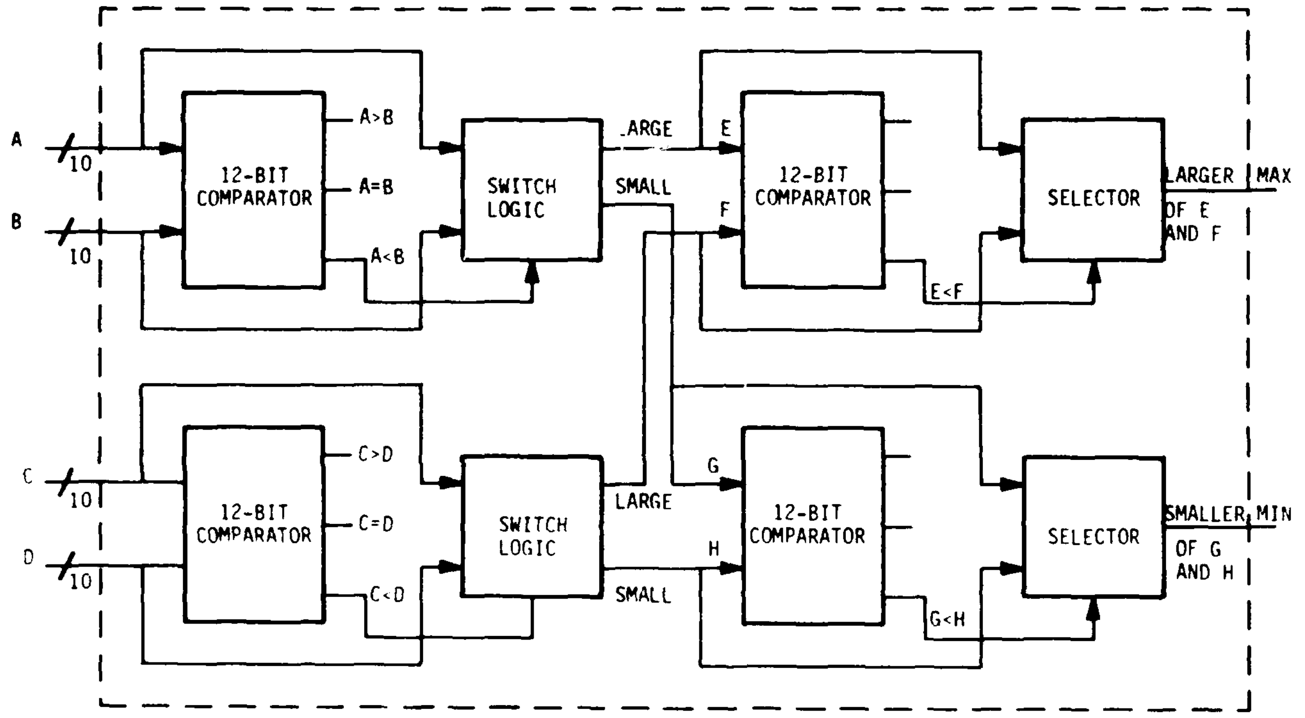


Figure 34b. Subdivide Pipeline Part II



*Logical "or" operation.

Figure 35. Grid Point Test



NOTE: A = X_{00} or Y_{00}, B = X_{01} or Y_{01}, etc. (See Figure 35).

Figure 36. Max/Min Circuit as Part of Grid Point Test

The Z-buffer is a memory containing the distance to each displayed subpatch as well as its tone or intensity. Priority conflicts are resolved by comparing the distances to the two subpatches in question. The subpatch which is closest is entered into the Z-buffer and the other is discarded. Two Z-buffers allow readout of one frame to the display while the next frame is being assembled.

Thirty-two parallel channels are required for the Z-buffer. This estimate is based on the estimated requirement for approximately 2.3×10^6 intensity computations every frame. This results in an intensity computation every 14.5 nsec.

Assume that dense MOS memories will be used for these memory-intensive stages, with 180 nsec access times and 300 nsec cycle times. Consequently, the Z-buffer and intensity buffer updates will take

180 nsec	Read
40 nsec	Compare
300 nsec	Write
520 nsec	Total

The update speed required is 14.5 nsec/update. Thus, the number of memory channels needed is

$$\frac{520 \text{ nsec}}{14.5 \text{ nsec}} \simeq 36$$

Each of the 36 channels will address a specific sector of the display screen. Pixels will be assigned to one of the 36 channels on the basis of x, y, screen coordinates.

The intensity computation is a function of the angles between the LOS, the sun and the patch normal, and the texture function and an atmospheric attenuation term based on patch distance. The texture function is determined by table look-up, using the patch ground coordinates for addressing. The intensity computation pipeline is shown in Figure 37. This processor requires 29 multipliers, 18 adders, 76 registers, and two table look-up read-only memories (PROMS) per channel.

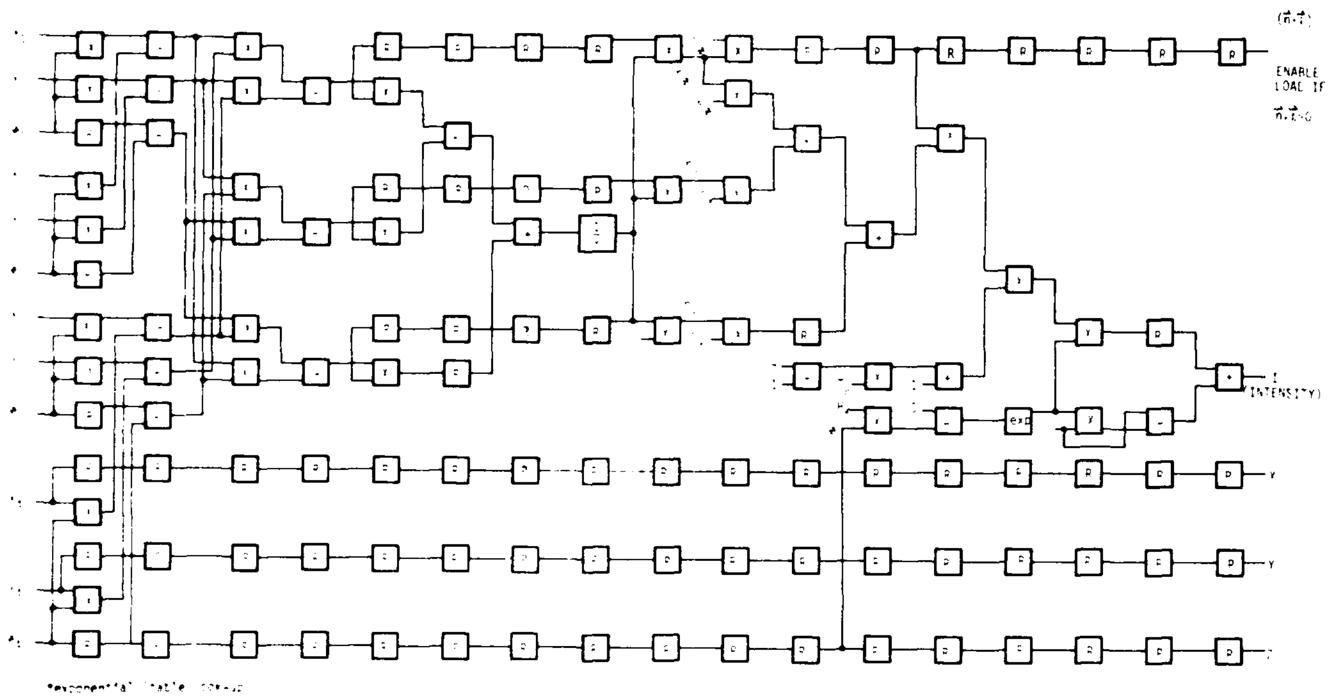


Figure 37. Compute Intensity Pipeline

Only seven channels of intensity computations are required. The intensity computation is pipelined and the pipeline clock rate is 10 MHz, so that the effective throughput rate/channel is 10^7.

The required throughput pixel rate is about $2.3 \times 3 \times 10^7$/second, resulting in a need for seven parallel channels. This is the product of the number of pixels/frame (10^6), the number of frames/second (30) and the additional factor (2.3) to account for extra computations from conflicts (Appendix C).

The total parts count for the system, excluding the host computer and mass memory, is contained in Table 1. This count also does not include control circuitry, multiplexers, etc., whose specification requires a more detailed design. Allowing 1000 devices for these additional functions, the total parts count is approximately 9000. Of this total, approximately two-thirds consist of memory devices, including 4000 64K x 1 devices for the Register Square Stack. As discussed previously, this estimate is probably conservative, but improvement requires further analysis by Monte Carlo simulation.

TIMING ANALYSIS

Figure 25 showed the throughput requirements in each stage of the pipelined processor in terms of the number of principal operations for each stage. These requirements are reflected in the subsequent estimation of the throughput and hence hardware requirements for each pipelined stage. The implication is that the preliminary architecture discussed supports the system throughputs shown in Figure 25. This subsection addresses the total transport delay in the pipelined cascaded stages.

The total transport delay is reckoned from the real time FOV memory to display. This measures the time elapsed from a new platform position input (from the flight simulator) to the display of this new position on the screen.

The transport delay T_D is the sum of the individual pipelined stage delays:

$$T_D = T_{REG} + T_{STACK} + T_{SUB} + T_{TEST} + T_{DISP}$$

TABLE 1. PARTS COUNT FOR PIPELINED APPROACH

Function	16 Bit Adders	16 Bit Subtractors	16 x 16 Bit Multipliers	16 x 16 Bit Dividers	16 Bit Latches	16 Bit Multistage Shift Registers	Memory (RAM)	Total Units
FOV Memory							32 (32K x 1)	32
Perspective Projection	2	3	3	1	3			13
Circular Buffer							12 (256 x 16)	12
Bicubic Fit	96		128		16			240
Register Square Computation	22				9	10		41
Register Square Stack							4000* (64K x 1)	4000*
Patch Subdivide (3 channels)	21/channel	10/channel			9/channel	32/channel		216
Grid Test (12 channels)	34/channel							408
Compute Intensity (7 channels)	35/channel	12/channel	30/channel		4/channel	9/channel		630
Z-Buffer Depth							40/channel (32K x 1)	1280
Intensity (32 channels)							32/channel (32K x 1)	1024
							Total	7896

*Upper limit based on one patch filling FOV.

where

	T_{REG}	= delay through register square computation,
	T_{STACK}	= delay through the stack,
	T_{SUB}	= delay through the patch subdivision,
	T_{TEST}	= delay through the tester,
and	T_{DISP}	= delay through the display processor

Each of these delays is estimated below.

$\underline{T_{REG}}$: Referring to Figure 26, this includes projection, circular buffer, bicubic fit and register square computation. The projection (Figure 28) requires five basic stages (subtract, multiply, two adds and a divide) of delay. Assuming 200 nsec of delay per stage (corresponding to the worst stage--the divide), this implies 200 x 5 = 1 μsec. The circular buffer adds a further delay of four rows of patches: 200 x 4 x 200 nsec = 160 μsecs. Here we have assumed 200 nsec/patch and 200 patches/row (corresponding to 40,000 patches in the FOV). The bicubic fit has a delay of six stages (Figure 30), or 6 x 200 nsec = 1.2 μsec. The register square pipeline (Figure 32) has a delay of four stages (800 nsec).

$$\text{Hence } T_{REG} = 1 + 160 + 1.2 + 0.8 = 163 \ \mu\text{secs}.$$

$\underline{T_{STACK}}$: This is basically the delay associated with the buffering of the patches, and is variable, depending on the size of the patches received (range to the patch). The worst case delay occurs when the stack fills up with patches and the succeeding pipelines are busy and cannot accept patches from the stack. This upperbound is 1 frame time (33 msec), i.e.,

$$T_{STACK} = 33 \text{ msec (max)}$$

$\underline{T_{SUB}}$: The subdivide pipeline entails a total delay of eight stages (see Figures 34a and 34b), or 1.6 μsecs at 200 nsec/stage

$$T_{SUB} = 1.6 \ \mu\text{secs}$$

$\underline{T_{TEST}}$: This consists of the gridpoint test (Figures 35 and 36) or eight stages at 200 nsecs/stage

Hence $T_{TEST} = 1.6\ \mu secs$

$\underline{T_{DISP}}$: The intensity computation pipeline involves (Figure 37) 17 stages at 200 nsecs, or

$$T_{DISP} = 3.4\ \mu secs$$

Hence, the total worst case transport delay

$$T_D = 163 + 33{,}000 + 1.6 + 1.6 + 3.4$$
$$= 33{,}169.6 \text{ nsecs}$$

Rounding off to the nearest larger frame time, the worst case transport delay

$$\boxed{T_D = 2 \text{ frame times}}$$

SECTION V

CONCLUSIONS AND RECOMMENDATIONS

The principal conclusion of this study is that real-time implementation of the approach is feasible for aircraft visual and sensor simulations. The hardware requirements are not unreasonable, although it seems that they could be significantly reduced through additional iterations of the design tradeoffs. In particular, the Register Square Stack member which was conservatively sized at 256 megabits, could be substantially reduced.

Improvements could be made in the test to determine whether a subpatch should be displayed. The number of patch subdivisions could be reduced to the point where perhaps only one Patch Subdivider pipeline would suffice if subdivision was not required to single picture element size.

Since the number of patches to be processed for fixed patch size increases as the square of the distance, it would be desirable to use variable patch sizes. Distant patches need not be as small as nearby patches where greater detail is necessary. This would not only provide variable resolution as a function of viewing distance but would reduce the size of the FOV Memory. Together with the previous suggestions for improving the grid test, this would also reduce the size of the Register Square Stack. The primary obstacle is the possibility of edge effects and artifacts where patches of different sizes are juxtaposed in the image plane.

Some hardware reduction may be possible in the Register Square Processor. With a fully pipelined design, the duty cycle of this stage is only 25 percent. Since it basically consists of a sequence of 4 x 4 matrix multiplications, some time-sharing of hardware may be possible. This stage is also a prime candidate for exploiting very large scale integrated circuit (VLSIC) technology.

Further work should be done on the Display Processor. The computation of tone or intensity, including the representation of texture, is by no means optimized at this point.

Finally, more work is required on anti-aliasing with this approach. Some anti-aliasing is inherent in the averaging of texture, based on projected patch size. However, this approach still exhibits severe aliasing problems along the horizon and along any boundary which separates a cultural object from a *terrain surface patch*.

An anti-aliasing approach has been under investigation at Honeywell for anti-aliasing with the bicubic subdivision algorithm. First, note that area sampling, i.e., averaging of patch contributions, is needed only at an *object's silhouette boundaries* where multiple surface contributions have to be averaged. This is because texture aliasing can be eliminated by prefiltering the texture pattern on a patch depending on its distance from the observer (farther patches are filtered more than the nearer ones). Therefore, point sampling the filtered texture can be used everywhere except at object or terrain silhouettes. In fact, this texture filtering is already performed implicitly in the current implementation of the Honeywell terrain representation program on STARS at AFHRL (by having several levels of texture detail and choosing the level corresponding to the patch distance). Hence, texture aliasing can be eliminated by adaptive texture filtering in real time before the patches are subdivided and tested.

Further, subpatches which come from silhouettes can be recognized by their surface normals (which are orthogonal to the line of sight). Hence, pixels which correspond to the silhouette edges can be treated separately from all the rest, and multiple surface averaging applied only at these pixels. Because the silhouette pixels comprise only a small fraction of the image, this insight results in a greatly simplified anti-aliasing algorithm for object space (depth buffer algorithms).

As explained above, texture prefiltering on each patch eliminates the need for area sampling everywhere but at the silhouette pixels. Hence, the unmodified point sampling version of the algorithm is applied to all patches (and subpatches) which are not tangential. However, tangential (silhouette) subpatches are treated differently. If a tangential subpatch is mapped to a certain pixel, the Z-buffer entry corresponding to the pixel is not compared or replaced with the new subpatch. Instead, a tangential subpatch arriving at a pixel signals that the point is a potential silhouette point. The current content of the Z-buffer is replaced with a pointer to a linked list which will contain information about the current and future contributions to that pixel. The current contents of the intensity buffer and Z-buffer for this pixel and the information from the new (silhouette) patch form as the first two elements of this list. At the end of the first pass, all nonsilhouette points are completely determined. Moreover, the information from all surfaces--the area (A), depth (Z) and intensity (I) is available in the linked lists for each potential silhouette pixel. This is then used to perform averaging and priority determination for the silhouette pixels.

The above approach satisfies the following goals for efficiency:

1. The expensive multi-surface area sampling is done only for the potential silhouette points, eliminating the jagged edges.

2. The use of linked lists for storing information on the silhouette points eliminates the need for multiple frame and Z-buffers, and uses only as much memory as needed to store the silhouette information.

3. Texture prefiltering enables the computationally simple (conventional) point sampling algorithm to be used almost everywhere in the image, without texture aliasing.

APPENDIX A

BICUBIC SPLINE INTERPOLATION

APPENDIX A

BICUBIC SPLINE INTERPOLATION

Currently, polygons are used to represent terrain elevation in computer-image generation. Approximation of terrain with polygons produces a faceted effect and a silhouette made of straight-line-segments. Curved surface segments or "patches" can be used instead of polygons. If the patches can be joined together with slope continuity, a picture of the surface can be made continuous both in shading and silhouette.

The most widely used nonlinear representation is the bicubic spline, an extension of the cubic spline interpolation for functions of one variable. In general, a spline curve is a piecewise analytical function whose pieces are chained together with continuity requirements on the first few derivatives imposed at the joints. The pieces are basic curve shapes with adjustable parameters for each piece. These parameters are chosen to satisfy the imposed continuity requirements. Polynomial curves are usually used for the pieces. The cubic polynomial $f(u) = au^3 + bu^2 + cu + d$ is the lowest degree polynomial with a sufficient number of parameters to provide continuous differentiability.

A curve in space can be represented in parametric form by the vector equation

$$\underline{r}(u) = \begin{bmatrix} x(u) \\ y(u) \\ z(u) \end{bmatrix}$$

where each of the components can be represented by a cubic defined for the parameter u in the interval. Usually, for convenience, $0 \le u \le 1$. In matrix notation, each of the cubics can be expressed as

$$f(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

A surface patch is a function of two variables u and v.

$$\underline{F}(u, v) = \begin{bmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{bmatrix}$$

Terrain elevation data is single-valued and is provided in the DMA data base as z(x, y) over a (nearly) uniform mesh x_i, y_i. By associating $x \rightarrow u$ and $y \rightarrow v$, the parametric representation of only the vertical component z(u, v) is needed.

The matrix notation for z(u, v) is

$$z(u, v) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

$$= \underline{u}\, \underset{\sim}{A}\, \underline{v}$$

where the a_{ij} are coefficients of the equation just as a, b, c, and d were in the univariate case. The problem is to find the coefficients.

Following Catmull[1], only local methods will be considered, i.e., each datum only affects the coefficients of nearby patches.

One method, using an array of 16 samples of z(u, v), might simultaneously solve the resulting linear system of equations in the unknown a_{ij}. This method would result in a bicubic patch which would pass through all the data points and interpolate the surface between. In addition to being computationally undesirable, this method would result in a surface that would be discontinuous in the first derivative at the patch boundaries.

Consider again the univariate equation. Four data samples, say f_1, f_2, f_3, f_4, are required to determine the four coefficients a, b, c, d. The coefficients can be related to the data samples by some 4 x 4 matrix $\underset{\sim}{M}$

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \underset{\sim}{M} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}$$

Therefore,

$$f(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \underset{\sim}{M} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}$$

This concept can be trivially extended to the bivariate case:

$$z(u,v) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \underset{\sim}{M} \begin{bmatrix} z_{11} & z_{12} & z_{13} & z_{14} \\ z_{21} & z_{22} & z_{23} & z_{24} \\ z_{31} & z_{32} & z_{33} & z_{34} \\ z_{41} & z_{42} & z_{43} & z_{44} \end{bmatrix} \underset{\sim}{M}^T \begin{bmatrix} v^3 & v^2 & v & 1 \end{bmatrix}^T$$

$$= \underline{u} \, \underset{\sim}{M} \, \underset{\sim}{Z} \, \underset{\sim}{M}^T \underline{v}$$

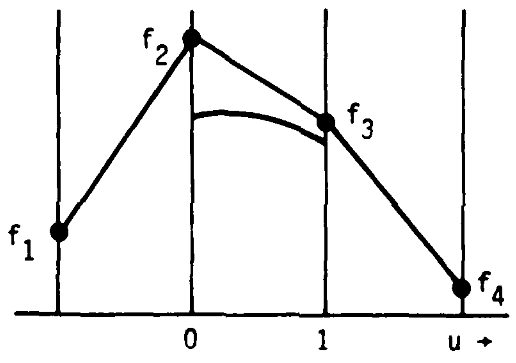
Note that $\underset{\sim}{A} = \underset{\sim}{M} \, \underset{\sim}{Z} \, \underset{\sim}{M}^T$. For the case considered earlier of fitting a bicubic through 16 points, the matrix M is given by

$$M_1 = \frac{1}{2} \begin{bmatrix} -9 & 27 & -27 & 9 \\ 18 & -45 & 36 & -9 \\ -11 & 18 & -9 & 2 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$

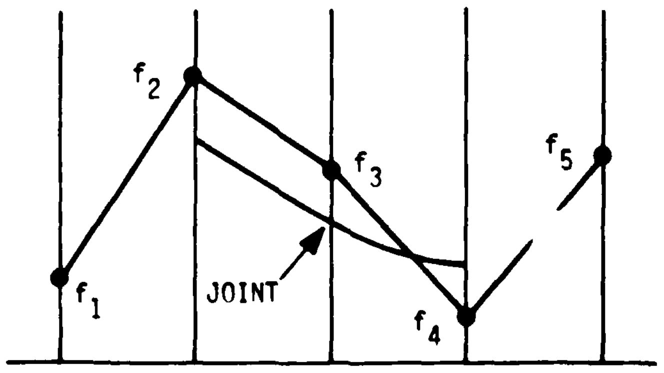
This technique circumvents the need to solve a system of 16 linear equations, but the problem of discontinuity in the derivative of the surface at the patch boundaries remains.

THE B-SPLINE

The cubic B-spline provides an alternative which provides continuity of the second derivative. In general, except when the points are collinear, the B-spline does not interpolate the data samples but rather approximates them. This property provides some smoothing of the input data samples. Consider the four points f_1, f_2, f_3, f_4:



A cubic curve segment can be generated which does not pass through any of the points. Similarly another segment of curve can be generated using the points f_2, f_3, f_4, and a fifth point f_5 which connects to the first segment with <u>second derivative</u> continuity.



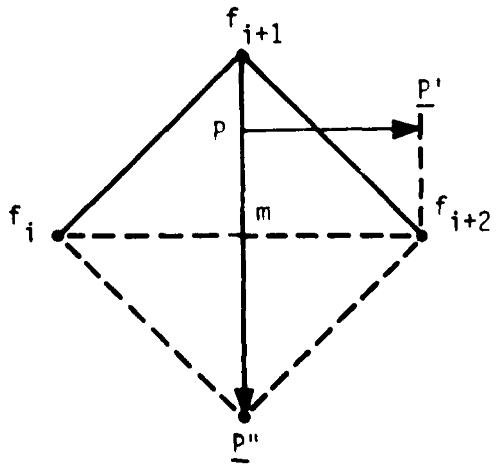
For the B-spline cubic, the matrix $\underset{\sim}{M}$ is derived from a set of basic functions. The weights of the basic functions are chosen to satisfy the continuity conditions, resulting in:

$$\underset{\sim}{M}_2 = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

There are some interesting geometric properties of these piecewise curves. They lie everywhere within the convex hull of the polygon of the defining vertices. At $u = 0$ or $u = 1$, the curve passes through a point P which is the 1/3 point of the median of the triangle formed by three sequential vertices. The first derivative vector at P is:

$$P' = \frac{f_{i+2} - f_i}{2}$$

and the second derivative vector at P is $P'' = (f_i - f_{i+1}) + (f_{i+2} - f_{i+1})$.

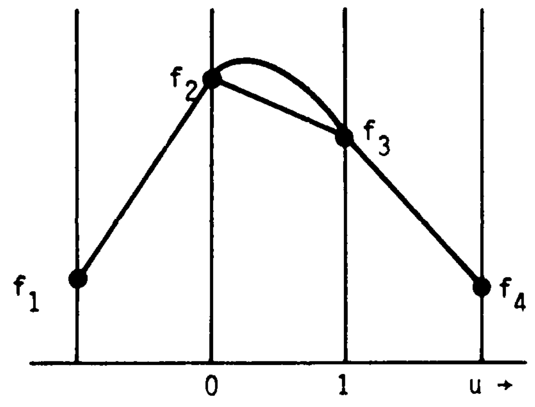


When the three vertices f_i, f_{i+1}, f_{1+2} are collinear, then the triangle is degenerate with P on the line. The tangent vector $\underline{P'}$ and second derivative $\underline{P''}$ are also along the line.

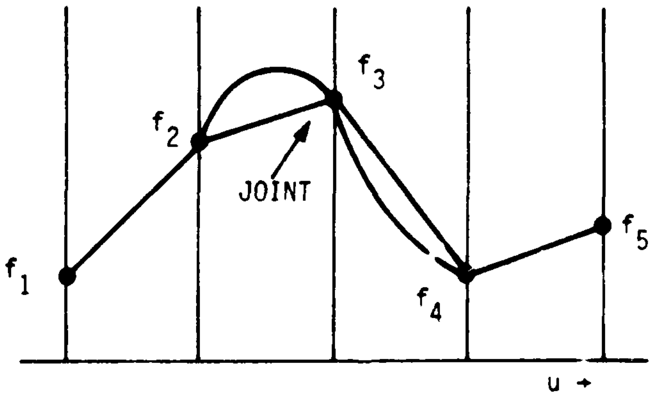
B-splines have the "variation diminishing" property which means that they approximate linear functions exactly, as demonstrated above, and the approximation is always "smoother" than the function it approximates.

THE CATMULL-ROM CUBIC SPLINE

This spline interpolates the data points and has continuity of <u>first derivative</u> at the joints. Consider the four points f_1, f_2, f_3, f_4:



A cubic can be generated that interpolates from point f_2 to f_3. Consider a fifth point f_5.



Another segment of curve is generated using f_2, f_3, f_4, f_5. The two segments join with first derivative continuity. For the Catmull-Rom spline, the matrix $\underset{\sim}{M}$ is

$$\underset{\sim}{M}_3 = \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$$

APPENDIX B

BICUBIC SPLINE SUBDIVISION

APPENDIX B

BICUBIC SPLINE SUBDIVISION

The binary bicubic spline subdivision algorithm described here is an adaptation of an algorithm developed by E. Catmull in his thesis published in December 1974 entitled, "A Subdivision Algorithm for Computer Display of Curved Surfaces." Catmull develops an algorithm for subdividing a cubic patch and its first derivatives in two orthogonal directions and discusses computational requirements for the algorithm.

CUBIC SUBDIVISION

Consider the cubic function

$$f(\tau) = a\,\tau^3 + b\,\tau^2 + c\,\tau + d$$

which is valid for $t - h \le \tau \le t + h$. By adding $f(t + h)$, the value of f at the center point is given by

$$\begin{aligned} f(t) &= [f(t - h) + f(t + h)]/2 = h^2\,(3at + b) \\ &= [f(t - h) + f(t + h)]/2 - g(t) \end{aligned} \qquad \text{(B-1)}$$

The correction term

$$g(t) = h^2\,(3at + b) \qquad \text{(B-2)}$$

can also be expressed in terms of its values at the ends of the interval

$$g(t) = [g(t - h) + g(t + h)]/2 \qquad \text{(B-3)}$$

Also note that, if h_n is the width of the n^{th} interval, at the $(n+1)^{st}$ binary subdivision $h_{n+1} = h_n/2$ and

$$g_{n+1}(\tau) = g_n(\tau)/4 \qquad \text{(B-4)}$$

Equations (B-1) through (B-4) provide the basis for the cubic spline subdivision algorithm described below. Using matrix algebra notations, define

$$\underline{c}_{n-1}(\tau) = \begin{bmatrix} f(\tau) \\ g_{n-1}(\tau) \end{bmatrix} \quad (2\text{x}1)$$

$$\underset{\sim}{Q} = \begin{Bmatrix} \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{Bmatrix} \quad (2\text{x}4)$$

$$\underset{\sim}{H} = \begin{Bmatrix} 1 & 0 \\ 0 & \frac{1}{4} \end{Bmatrix} \quad (2\text{x}2)$$

Then to scale the correction term for the subdivision in progress, form

$$\underline{c}_n(\tau) = \underset{\sim}{H}\ \underline{c}_{n-1}(\tau) = \begin{bmatrix} f(\tau) \\ g_{n-1}(\tau)/4 \end{bmatrix} = \begin{bmatrix} f(\tau) \\ g_n(\tau) \end{bmatrix}$$

To compute the function and correction term for the center point of the interval, $\tau_1 \leq \tau \leq \tau_2$,

$$\underline{c}_n\left(\frac{\tau_1 + \tau_2}{2}\right) = \underset{\sim}{Q} \begin{bmatrix} \underline{c}_n(\tau_1) \\ \underline{c}_n(\tau_2) \end{bmatrix} = \begin{bmatrix} [f(\tau_1) + f(\tau_2)]/2 - [g_n(\tau_1) + g_n(\tau_2)]/2 \\ [g_n(\tau_1) + g_n(\tau_2)]/2 \end{bmatrix} \quad (\text{B-5})$$

Because the elements of $\underset{\sim}{H}$ and $\underset{\sim}{Q}$ are powers of 1/2, a binary search of a cubic spline is computationally reduced to shifts and adds. This desirable property carries over into subdivision of a bicubic spline function.

BICUBIC SUBDIVISION

As described in Appendix A, the bicubic spline for a terrain patch can be described by the surface

$$z(u, v) = \underline{u}' \underset{\sim}{M} \underset{\sim}{Z} \underset{\sim}{M}' \underline{v} \qquad \text{(B-6)}$$

where $\underline{u}' = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}; \quad 0 \le u \le 1$

$\underline{v}' = \begin{bmatrix} v^3 & v^2 & v & 1 \end{bmatrix}; \quad 0 \le v \le 1$

(u, v) are independent variables (corresponding to the orthogonal coordinates (x, y) for the bicubic surface within the patch; the prime denotes a transpose. $\underset{\sim}{M}$ is a (4 x 4) matrix of second derivatives, and $\underset{\sim}{Z}$ is a (4 x 4) matrix of surface elevations surrounding and including the patch in question, i. e.,

$$\underset{\sim}{Z} = \{z_{ij}\} \equiv \{z(x_i, y_j)\}$$

where, if $1 \le i$ and $i \le 4$, the terrain patch in question is bounded by

$$x_2 \le x \le x_3$$

$$y_2 \le y \le y_3.$$

To develop the extension of the subdivision algorithm to two dimensions, define

$$\underset{\sim}{A} = M\, Z\, M' \equiv \{a_{ij}\} \; (4x4)$$

and consider the vector

$$\underline{s}'(u) = \underline{u}' \underset{\sim}{A} \equiv [s_1 \, s_2 \, s_3 \, s_4]$$

with

$$s_j = \sum_{i=1}^{4} u^{(4-i)} a_{ij}.$$

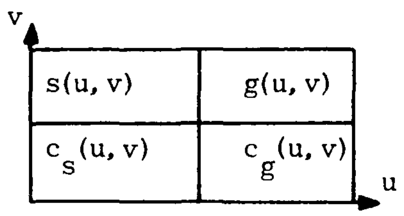
Then

$$s(u, v) = \sum_{j=1}^{4} v^{(4-j)} s_j \qquad \text{(B-7)}$$

For constant v, each s_j is a cubic in u which has a correction term g_j associated with it. In particular, $s(u, v_o)$ has the correction term

$$g(u, v) = \sum_{j=1}^{4} v_o^{(4-j)} g_j(u) \tag{B-8}$$

For constant u, both $s(u_o, v)$ and $g(u_o, v)$ are cubics in v. Hence each has its own correction term, say $c_s(u_o, v)$ for s and $c_g(u_o, v)$ for g. Following Catmull, these four functions can be arranged in a "register square:"



Moving in the v direction, c_s corrects s, and c_g corrects g. Conversely, in the u direction, g corrects s and c_g corrects c_s. Based on the cubic subdivision algorithm described previously, these functions are explicitly

$s(u, v) = \sum_{j=1}^{4} v^{(4-j)} \sum_{i=1}^{4} u^{(4-i)} a_{ij}$	$g(u, v) = h_u^2 \sum_{j=1}^{4} v^{(4-j)} (3ua_{1j} + a_{2j})$
$c_s(u, v) = h_v^2 \sum_{i=1}^{4} u^{(4-i)} [3va_{i1} + a_{i2}]$	$c_g(u, v) = h_u^2 h_v^2 \left[3v(3ua_{11} + a_{21}) + (3ua_{12} + a_{22})\right]$

In particular, the register squares corresponding to the corner of the terrain patch prior to subdivision ($h_u = h_v = 1$) are given by:

(0, 1)	$\sum_{j=1}^{4} a_{4j}$	$\sum_{j=1}^{4} a_{2j}$	$\sum_{i=j}^{4} \sum_{j=1}^{4} a_{ij}$	$\sum_{j=1}^{4} (3a_{1j} + a_{2j})$	(1, 1)
	$3a_{41} + a_{42}$	$3a_{21} + a_{22}$	$\sum_{i=1}^{4} (3a_{i1} + a_{i2})$	$3(3a_{11} + a_{21}) + (3a_{12} + a_{22})$	
	a_{44}	a_{24}	$\sum_{i=1}^{4} a_{i4}$	$3a_{14} + a_{24}$	
(0, 0)	a_{42}	a_{22}	$\sum_{i=1}^{4} a_{i2}$	$3a_{12} + a_{22}$	(1, 0)

With these preliminary definitions, the bicubic subdivision algorithm can be expressed in matrix notation as follows. For any patch or subpatch, let the register squares for the corners of the patch be described by a set of (2 x 2) matrices in the form

$\underset{\sim}{C}_{01}^{(n-1)}$	$\underset{\sim}{C}_{11}^{(n-1)}$
$\underset{\sim}{C}_{00}^{(n-1)}$	$\underset{\sim}{C}_{10}^{(n-1)}$

In preparation for subdivision, the correction terms in each register square are first scaled by the operator

$$\underset{\sim}{H} = \begin{Bmatrix} 1 & 0 \\ 0 & \frac{1}{4} \end{Bmatrix}$$

to produce scaled register squares

$$\underset{\sim}{C}^{(n)} = \underset{\sim}{H}\, \underset{\sim}{C}_{k\ell}^{(n-1)}\, \underset{\sim}{H}$$

Subdivision will produce a (3 x 3) block of (2 x 2) register square matrices of the form

$\underset{\sim}{C}^{(n)}_{01}$	$\underset{\sim}{C}^{(n)}_{\frac{1}{2},1}$	$\underset{\sim}{C}^{(n)}_{11}$
$\underset{\sim}{C}^{(n)}_{0,\frac{1}{2}}$	$\underset{\sim}{C}^{(n)}_{\frac{1}{2},\frac{1}{2}}$	$\underset{\sim}{C}^{(n)}_{1,\frac{1}{2}}$
$\underset{\sim}{C}^{(n)}_{00}$	$\underset{\sim}{C}^{(n)}_{\frac{1}{2},0}$	$\underset{\sim}{C}^{(n)}_{10}$

In terms of the subdivision operator,

$$\underset{\sim}{Q} = \left\{ \begin{matrix} \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{matrix} \right\}$$

the new register squares resulting from the bicubic subdivision are then given by

$$\underset{\sim}{C}^{(n)}_{\frac{1}{2},0} = \left\{ \underset{\sim}{C}^{(n)}_{00} \quad \underset{\sim}{C}^{(n)}_{10} \right\} \underset{\sim}{Q}' \; ; \; \underset{\sim}{C}^{(n)}_{\frac{1}{2},1} = \left\{ \underset{\sim}{C}^{(n)}_{01} \quad \underset{\sim}{C}^{(n)}_{11} \right\} \underset{\sim}{Q}'$$

$$\underset{\sim}{C}^{(n)}_{0,\frac{1}{2}} = \underset{\sim}{Q} \left\{ \begin{matrix} \underset{\sim}{C}^{(n)}_{01} \\ \underset{\sim}{C}^{(n)}_{00} \end{matrix} \right\} \; ; \; \underset{\sim}{C}^{(n)}_{1,\frac{1}{2}} = \underset{\sim}{Q} \left\{ \begin{matrix} \underset{\sim}{C}^{(n)}_{11} \\ \underset{\sim}{C}^{(n)}_{10} \end{matrix} \right\}$$

$$\underset{\sim}{C}^{(n)}_{\frac{1}{2},\frac{1}{2}} = \left\{ \underset{\sim}{C}^{(n)}_{0,\frac{1}{2}} \quad \underset{\sim}{C}^{(n)}_{1,\frac{1}{2}} \right\} \; ; \; \underset{\sim}{Q}' = \underset{\sim}{Q} \left\{ \begin{matrix} \underset{\sim}{C}^{(n)}_{\frac{1}{2},1} \\ \underset{\sim}{C}^{(n)}_{\frac{1}{2},0} \end{matrix} \right\}$$

Any of the four resulting distinct sets of four register squares can then be subdivided by repretion of the algorithm.

To close with an example, suppose $a_{4j} = j$ and $a_{2j} = (5-j)$. Then

$$\underset{\sim}{C}_{01}^{(0)} = \boxed{\begin{array}{c|c} 10 & 10 \\ \hline 5 & 15 \end{array}}, \quad \underset{\sim}{C}_{00}^{(0)} = \boxed{\begin{array}{c|c} 4 & 1 \\ \hline 2 & 3 \end{array}}$$

$$\underset{\sim}{C}_{01}^{(1)} = \boxed{\begin{array}{c|c} 10 & 5/2 \\ \hline 5/4 & 15/16 \end{array}}, \quad \underset{\sim}{C}_{00}^{(1)} = \boxed{\begin{array}{c|c} 4 & 1/4 \\ \hline 1/2 & 3/16 \end{array}}$$

$$\underset{\sim}{C}_{0,\frac{1}{2}}^{(1)} = \boxed{\begin{array}{c|c} 49/8 & 13/16 \\ \hline 7/8 & 9/16 \end{array}}$$

It is worth noting that the off-diagonal correction terms in C decrease as 4^{-n}, and the diagonal correction term decreases as 16^{-n}. This fact can perhaps be used to reduce the computational load by easing into a linear interpolation (i.e., stop updates of the correction terms) after a few subdivisions in the binary search have been performed.

Surface Derivatives

In addition to searching the surface of the bicubic patch for an intersection with the LOS, the first derivatives of the surface are also required to compute the direction cosines of the surface normal at the point of intersection and thereby the reflected intensity for a given sun angle.

If one considers the cubic and its derivative

$$f(\tau) = a\,\tau^3 + b\,\tau^2 + c\,\tau + d$$

$$f_\tau(\tau) = 3a\tau^2 + 2b\tau + c$$

the derivative at the center of the interval $t - h \leq \tau \leq t + h$ can be expressed as either

$$f_T(t) = [f(t+h) = f(t-h)] / (2h) - ah^2$$

or

$$f_T(t) = [f_T(t-h) + f_T(t+h)] / 2 - 3ah^2 \qquad \text{(B-9)}$$

Catmull suggests the former approach, but the latter is more attractive for the two-dimensional subdivision algorithm because the same subdivision algorithm can be used for the surface derivatives as for the surface. More importantly, the algorithm based on the first equation requires that the four patches bounding the patch in question be subdivided simultaneously. For example, along an outer boundary such as u = 0, computation of s_u (0, v) requires knowledge of $s(-h_n, v)$, which is not contained in the patch being subdivided.

Proceeding as in the case of the surface subdivision, and using Equation (B-9) as the basic one-dimensional algorithm, consider the first derivative in the u direction:

$$s_u(u,v) = [3u^2 \ 2u \ 1 \ 0] \underset{\sim}{A} \underline{v} = \sum_{j=1}^{4} v^{(4-j)} \sum_{i=1}^{3} (4-i)\, u^{(3-i)} a_{ij} .$$

For constant v, $s_u\,(u, v_o)$ is a quadratic in u with correction term

$$g_u(u, v_o) = 3h_u^2 \sum_{j=1}^{4} v_o^{(4-j)} a_{1j}$$

On the other hand, for constant u, both $s_u(u_o, v)$ and $g_u(u_o, v)$ are cubics in v and have the appropriate cubic form for the correction coefficients.

Based on these observations, the initial set of register squares for the u derivative of the terrain patch can be quickly determined.

Register Squares for $s_u(u, v)$ Patch

(0, 1) (1, 1)

$\sum_{j=1}^{4} a_{3j}$	$3 \sum_{j=1}^{4} a_{1j}$	$\sum_{j=1}^{4} \sum_{i=1}^{3} (4-1)a_{ij}$	$3 \sum_{j=1}^{4} a_{1j}$
$3a_{31} + a_{32}$	$3(3a_{11} + a_{12})$	$\sum_{i=1}^{3} (4-i)(3a_{i1} + a_{i2})$	$3(3a_{11} + a_{i2})$
a_{34}	$3a_{14}$	$\sum_{i=1}^{3} (4-i)a_{i4}$	$3a_{14}$
a_{32}	$3a_{12}$	$\sum_{i=1}^{3} (4-i)a_{i2}$	$3a_{12}$

(0, 0) (1, 0)

A similar consideration of the v derivative of the surface, $s_v(u, v)$, yields a set of register squares which are similar to those for $s_u(u, v)$ but with the (i, j) indices interchanged, the $\underset{\sim}{C}_{k\ell}$ transposed, and the diagonal $\underset{\sim}{C}$ interchanged.

Register Squares for $s_v(u, v)$ Patch

(0, 1) (1, 1)

$\sum_{j=1}^{3} (4-j)a_{4j}$	$\sum_{j=1}^{3} (4-j)a_{2j}$	$\sum_{i=1}^{4} \sum_{j=1}^{3} (4-j)a_{ij}$	$\sum_{j=1}^{3} (4-j)(3a_{1j} + a_{2j})$
$3a_{41}$	$3a_{21}$	$3\sum_{i=1}^{4} a_{i1}$	$3(3a_{11} + a_{21})$
a_{43}	a_{23}	$\sum_{i=1}^{4} a_{i3}$	$3a_{13} + a_{23}$
$3a_{41}$	$3a_{21}$	$3\sum_{i=1}^{4} a_{i1}$	$3(3a_{11} + a_{21})$

(0, 0) (1, 0)

APPENDIX C

PROJECTED AREA OF A PATCH AT THE SCREEN

APPENDIX C

PROJECTED AREA OF A PATCH AT THE SCREEN

The first step is to derive a key relationship between the orientation of a patch, relative position of the observer, and the projected area of a patch at the screen. Much of the further analysis is based on this relationship.

The imaging geometry is shown in Figure C-1. The patch is approximately at a distance η from the observer (who is in the ζ-η plane). The observer is at height ζ relative to the patch. The normal to the patch makes an angle θ with the vertical, and its projection on the $\zeta\eta$ plane makes an angle ϕ with the ξ axis (cylindrical notation). The line-of-sight makes an angle β with the horizontal. The patch size is h x h.

For a flat terrain, the projected area of the patch in angular subtense at the screen is given by:

$$\text{Area} = \frac{h^2}{\eta^2} \cos A \left(\text{rad}^2 \right) \qquad \text{(C-1)}$$

where A is the angle made by the line-of-sight vector $\vec{l}$ with the patch normal $\vec{n}$.

Now, $\cos A = \vec{l} \cdot \vec{n}$ where:

$$\vec{l} = [\cos \beta,\ 0,\ \sin \beta]$$

and

$$\vec{n} = [\sin \theta \cos \phi,\ \sin \theta \sin \phi,\ \cos \theta]^T$$

Then:

$$\cos A = [\sin \beta \cos \theta + \cos \beta \sin \theta \cos \phi]. \qquad \text{(C-2)}$$

Substituting (C-2 into C-1):

$$\text{Area} = \frac{h^2}{\eta^2} [\sin \beta \cos \theta + \cos \beta \sin \theta \cos \phi] \qquad \text{(C-3)}$$

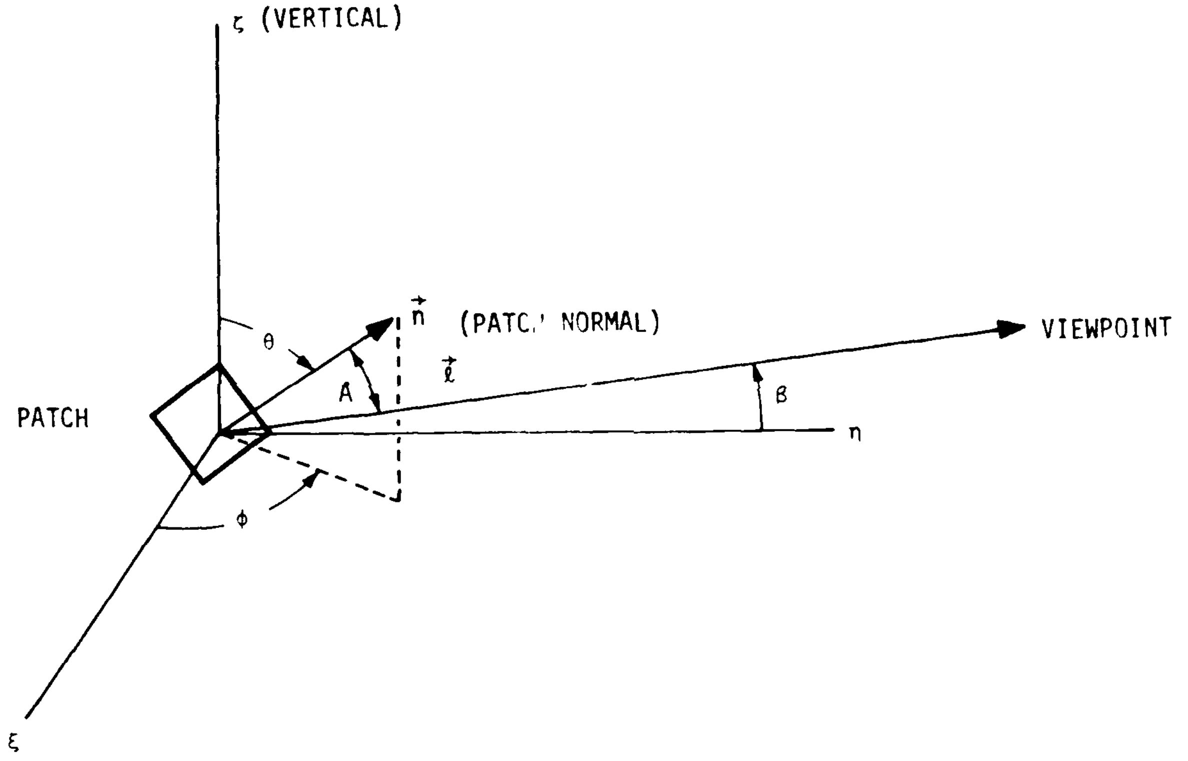


Figure C-1. The Projected Area of a Patch at the Screen

For small β, the observation angle can be approximated by:

$$\sin \beta \simeq \zeta/\eta$$

$$\cos \beta \simeq 1$$

Similarly, $\sin \theta \simeq \theta$ and $\cos \theta \simeq 1$ for small θ.

Hence the area of the patch becomes:

$$\text{Area} \simeq \frac{h^2}{\eta^2} \left[\frac{\zeta}{\eta} + \theta \cos \phi \right] \left(\text{rad}^2 \right) \qquad \text{(C-4)}$$

Equation (C-4) gives the projected area of a patch at a specific tilt orientation (θ, ϕ), distance η, and relative height ζ from the observer. In practice, of course, patches can have different tilts θ, orientations ϕ, and heights ζ with respect to the observer. Hence, to find the average projected area of a patch at a given distance, statistical models of the parameters θ, ϕ, and ζ have to be generated. This statistical model gives rise to a means of characterizing the terrain roughness.

TERRAIN ROUGHNESS MODEL

The terrain is composed of square patches (of side h) each with an angle θ to the vertical, an angle ϕ to the vertical plane containing the patch and the observer, and height ζ below the observer (Figure C-1). Note that θ denotes whether a patch is tilted, and ϕ determines whether it is tilted toward or away from the observer. The following assumptions can be made about the distribution of θ, ϕ and ζ over the entire field of view:

- θ, ϕ and ζ are independent random variables
- θ, ϕ and ζ are stationary processes in the field of view--that is, the distribution of the parameters is not a function of the location.

It can be further assumed that ϕ is uniformly distributed between 0 and 2π. This means it is equally likely that the observer is looking at a patch from any direction of the compass. The distribution of θ, the tilt of a patch to the vertical, need not be completely specified. It is completely characterized for purposes of this analysis by its mean value θ_m. Finally, ζ is assumed to be a normal random variable with mean m_ζ and standard deviation σ_ζ. m_ζ and σ_ζ are measured over the significant region of the field of view.

The terrain roughness is then completely characterized by θm, the average slope of a patch, and σ_ζ, the standard deviation of the patch elevations over a small area.

THE AVERAGE AREA OF A PATCH AT DISTANCE X

From the terrain roughness model, the average area of a patch in equation (C-4) is computed by integrating over the distributions of θ, ϕ and ζ. Since these parameters are assumed to be independent, the order of integration is immaterial.

The result in equation (C-4) is first averaged over all possible orientations of the observer with respect to the patch. Since each orientation ϕ is equally likely, the probability density of ϕ is:

$$p(\phi) = \begin{cases} \frac{1}{2\pi} & 0 < \phi < 2\pi \\ 0 & \text{elsewhere} \end{cases} \qquad \text{(C-5)}$$

Since the "back" of a patch must be subdivided if it is visible, negative areas (of a patch facing away from observer) should be counted as positive areas. Therefore, the average over ϕ should be written:

$$E_\phi[\text{area}] = \frac{h^2}{\eta^2} \, E \left[\frac{\zeta}{\eta} + \theta \cos \phi \right] \qquad \text{(C-6)}$$

where $| \cdot |$ denotes absolute value, and E [ ] denotes expected value or average.

Note that as ϕ varies from 0 to π, cos ϕ takes on values from between +1 and -1. Therefore, the above quantity is difficult to evaluate in closed form unless the specific value of ζ/η is known. Instead, Schwartz's inequality gives:

$$E_\phi\,[\text{area}] \leq \frac{h^2}{\eta^2} \left[\left| \frac{\zeta}{\eta} \right| + E_\phi \left[\theta \cos \phi \right] \right] \qquad \text{(C-7)}$$

which gives an upper bound (conservative estimate) on the area.

Now: $$\theta\, E_{\phi}\left[\cos\phi\right] = \theta \cdot \frac{2}{2\pi}\int_{-\pi/2}^{\pi/2} \cos\phi \, d\phi \qquad \text{(C-8)}$$

$$= \frac{2}{\pi}\,\theta$$

Hence: $$E_{\phi}\,[\text{area}] = \frac{h^2}{\eta^3}\,|\zeta| + \frac{h^2}{\eta^2}\cdot\frac{2}{\pi}\,\theta\left(\text{rad}^2\right) \qquad \text{(C-9)}$$

Since this equation is linear in θ, averaging over all possible θ and assuming θ has a mean value of θ_m gives:

$$E_{\phi}\,[\text{area}] = \frac{h^2}{\eta^3}\,|\zeta| + \frac{h^2}{\eta^2}\cdot\frac{2}{\pi}\,\theta_m\left(\text{rad}^2\right) = A \qquad \text{(C-10)}$$

Note that the first term in the above equation corresponds to the flat earth case. The second term modifies the result for a rough terrain.

DISTRIBUTION OF PATCH PROJECTED AREAS

Equation (C-10) gave the average projected area of a patch at a distance η and relative height ζ from the sensor. To compute the total projected area at the screen due to all the patches in the FOV, the distribution of the patch distances must be determined--that is, how many patches are at a given distance from the observer. The number of patches in the strip $d\eta$ at a distance η is given by:

$$p(\eta)d\eta = \frac{\eta}{h^2}\,\psi_H\, d\eta \qquad \text{(C-11)}$$

The total projected area is obtained by integrating (C-10) and (C-11) over the near and far distances in the FOV, that is:

$$A(\ell_1, \ell_2) = \int_{\ell_1}^{\ell_2} A(\eta)p(\eta)d\eta$$

$$= \int_{\ell_1}^{\ell_2} \left[\frac{\zeta}{\eta} \cdot \psi_H + \frac{2}{\pi} \theta_m \frac{\psi_H}{\eta} \right] d\eta$$

$$= \zeta \psi_H \left[\frac{\ell_2 - \ell_1}{\ell_1 \ell_2} \right] + \frac{2}{\pi} \theta_m \psi_H \text{ Ln} \left(\frac{\ell_2}{\ell_1} \right) \left| \text{rad}^2 \right| \qquad \text{(C-12)}$$

Because $\ell_2 >> \ell_1$, equation (C-12) can be written as:

$$A(\ell_1, \ell_2) = \frac{\zeta}{\ell_1} \psi_H + \frac{2}{\pi} \theta_m \psi_H \text{ Ln} \left(\frac{\ell_2}{\ell_1} \right) \qquad \text{(C-13)}$$

The projected area is, of course, a function of the near distance ℓ_1 (the distance to the bottom of the FOV). The worst case is when the horizon is at the top of the screen as shown in Figure C-2. Then the vertical field of view determines ℓ_1 because from Figure C-2, $\psi_V = \zeta/\ell_1$.

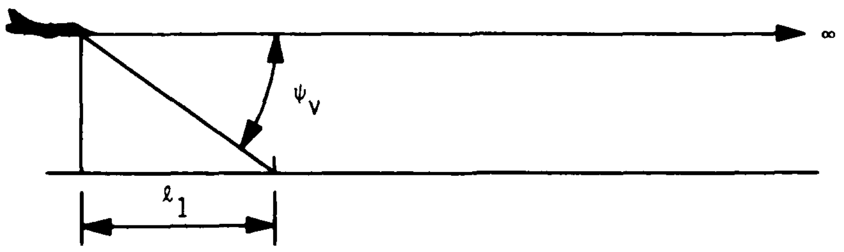


Figure C-2. Imaging Geometry for the Worst Case

Then equation (C-13) becomes:

$$A = \psi_v \psi_H \left[1 + \frac{2}{\pi} \cdot \frac{\theta_m}{\psi_v} \cdot \mathrm{Ln} \left(\frac{\ell_2}{\ell_1} \right) \right] \quad \text{(C-14)}$$

Assuming an nxn element raster over the FOV ($\psi_v \cdot \psi_H$), the area in number of raster elements is given by:

$$A = n^2 \left[1 + \frac{2}{\pi} \cdot \frac{\theta_m}{\psi_v} \cdot \mathrm{Ln} \left(\frac{\ell_2}{\ell_1} \right) \right] \quad \text{(C-15)}$$

Equation (C-15) explicitly shows the total projected area for a rough terrain when the entire field of view is covered with the ground patches. In the flat earth case, it is obvious that $A = n^2$ which is intuitively correct. With rough terrain ($\theta_m > 0$), because of hidden surfaces more than one point on the ground maps to the same point on the screen. This is the reason why the total projected area can be greater than n^2 raster elements. To see how rough terrain affects the quantity in equation (C-15), assume $\ell_1 \simeq 500$ m, $\ell_2 = 20$ km, and a vertical field of view $\psi_v = 30^\circ$. Table C-1 shows the total projected area as a function of θ_m, the roughness parameter.

Table C-1 shows that for $\theta_m = 15$ (extremely rough mountain terrain) the total projected area is 2.27 times the corresponding result for the flat earth case because of hidden surfaces.

TABLE C-1. AREA VS. θ_m

θ_m°	A/n^2
0	1
2	1.16
5	1.39
7.5	1.59
10	1.78
12	1.94
15	2.27

APPENDIX D

NUMBER OF PATCHES NEEDING BICUBIC SUBDIVISION

APPENDIX D

NUMBER OF PATCHES NEEDING BICUBIC SUBDIVISION

As discussed earlier, the number of potentially visible patches in the field of view is very large (200,000 to 400,000 to a range of 50 miles). However, not all these patches have to be subdivided to render them on the display. Whether or not a patch has to be subdivided depends on its projected area in terms of the number of screen grid crossings it covers. If it covers more than one screen grid intersection, it has to be subdivided. Furthermore, as mentioned in the discussion of the project and subdivide approach, it may be desirable to use the bicubic subdivision algorithm on only the nearer-in patches which span a large number of screen grid intersections. More distant patches may be subdivided using a linear subdivision. This saves the expensive real-time bicubic fit and register square computation on the large number of patches which can be subdivided using linear subdivision. Here, the relationship is quantified between the number of patches which require bicubic subdivision and the sensor, terrain, and flight parameters which affect this number.

Assume that a patch will be subdivided using bicubic subdivision if its projected area (in terms of grid intersections) is greater than m. Let ℓ_m be the distance at which a patch subtends on the average, m picture grid points. The number of patches in the field of view up to ℓ_m is given by the equation to be:

$$B_m = \frac{\psi_H \ell_m^2}{2h^2} \qquad \text{(D-1)}$$

The screen grid intersections subtended by a patch is a function of the resolution of the sensor and the tilt of the patch with respect to the observer and the distance to the observer. In Appendix C, the average projected area (in angular subtense) of a patch was:

$$A = \frac{h^2}{r^3} \, |\zeta| + \frac{h^2}{r^2} \cdot \frac{2}{\pi} \cdot \theta_m \left(\text{rad}^2\right) \qquad \text{(C-10)}$$

Where: h is the size of the patch;
η is the distance to the observer;
ζ is the height of the observer relative to the patch; and
θ_m is the "average" slope of the patch with respect to the horizontal, a function of the terrain roughness.

At large distances and rough terrain, the second term in Equation (C-10) dominates the first term, that is:

$$A \simeq \frac{h^2}{\eta^2} \cdot \frac{2}{\pi} \cdot \theta_m \left|\text{rad}^2\right| \tag{D-2}$$

The number of screen grid intersections is given by:

$$m = \frac{A}{d\psi^2} \tag{D-3}$$

Where $d\psi$ is the resolution of the simulated sensor. The distance ℓ_m at which a patch subtends m screen grid intersections is given by equations (D-2) and (D-3) as:

$$m = \frac{1}{d\psi^2} \cdot \frac{h^2}{\ell_m^2} \cdot \frac{2}{\pi} \theta_m \tag{D-4}$$

From which:

$$\ell_m = \frac{h}{d\psi} \sqrt{\frac{2\,\theta_m}{m\pi}} \tag{D-5}$$

The number of patches B_m in the field of view at a distance less than ℓ_m is given by (D-1) and (D-4) to be:

$$B_m = \frac{\psi_H}{d\psi^2} \cdot \frac{\theta_m}{\pi m} \tag{D-6}$$

If $\psi_H/d\psi = n$, (the number of resolution elements in a line of the display), Equation (D-6) can be rewritten as:

$$B_m = \frac{1}{m\pi} \left(\frac{\theta_m}{\psi_H} \right) n^2 \qquad \text{(D-7)}$$

The result in Equation (D-7) is interesting because it explicitly shows the dependence of B_m on the surface roughness (θ_m), horizontal field of view (ψ_H), and number of display elements (n^2). For a sensor with $\psi_H = 60^\circ$, and assuming a typical rough terrain of $\theta_m = 10^\circ$, $B_m \simeq 212000/m$ is derived from Equation (D-7). If the threshold m = 5, only 21200 patches have to be subdivided using the bicubic subdivision algorithm; that is, the bicubic fits and register squares must be computed in real time on these patches. The remaining patches in the field of view can be subdivided linearly.

At this time, no subjective simulation results have been obtained as to the largest value of m which will not produce artifacts due to the linear interpolation. A factor m = 5 appears reasonable, especially in view of the conservative 1000 x 1000 display resolution assumed.

☆U.S. GOVERNMENT PRINTING OFFICE: 1981- 771-055/108